CHAPTER

2

How Software and Hardware Work Together

In this chapter, you will learn:

- What happens when you first turn on a computer, so that both the hardware and the software are poised to follow your directions
- How hardware interacts with the system and how software manages hardware resources
- Practical and easy ways to protect hardware and software

In Chapter 1, you experienced "the grand tour" of all hardware and software components that make up a computer system. You saw hardware devices used for input, output, processing, and storage of data. You saw the hardware components that make up the electrical system of the computer and the components used for communicating both data and instructions from one device to another—buses on the system board being of paramount importance in that communication. The CPU was hailed as the central processing point for all data and instructions, and you learned that both must be stored in memory with assigned memory addresses before processing can begin.

On the software side, you learned that software works in layers, with the lowest layer (BIOS and device drivers) interfacing with hardware, and the highest layer (applications software) interfacing with the user. The OS is the middleman layer coordinating everything.

Chapter 1 surveyed individual components that make up a computer system. The focus of Chapter 2 is on learning how these individual components work together to perform tasks. If Chapter 1 is the grand tour, then this chapter invites you backstage to see firsthand how things work. First, the chapter continues with the boot process presented in Chapter 1. You will study booting in detail, and learn how you can customize the process, and troubleshoot and solve boot problems.

The boot process will serve as an introduction to thoroughly understanding how hardware and software components work together, which is the heart of this chapter. The major goal of this chapter is that, at its conclusion, you will say, "Aha, I finally understand the basics of how a computer really works!" Essential to this mastery is understanding how lines on a system-board bus are used, how hardware makes requests of the CPU to process its data, and how software, using the CPU, passes commands and requests for information to a hardware device. Before studying how this interaction takes place, we first look at the tools components use to do all that. These tools, including memory and other shared resources that peripheral hardware and software use for interaction, are both physical and logical; they're called system resources and are surveyed early in our discussion.

Our discussion of how computers work, presented in this chapter and Chapter 1, would be incomplete if we omitted a discussion of one of the most fundamental tasks of managing computer resources: protecting data, software, and hardware from the most common hazards. This chapter ends with a section describing how to make and use backups, how to protect a computer from hazardous electricity, and the practical precautions to take when troubleshooting a computer problem.

The Boot, or Startup, Process

Remember that computer hardware without software is about as useless as a car without a driver. Recall from Chapter 1 that the mechanic and chauffeur shown in Figure 2-1 were compared to startup software that prepared the hardware for action and other software that controlled the hardware once the startup process was completed. The user is like the passenger who directs the chauffeur. In this chapter, the analogy goes a step further. Software controls hardware by way of the CPU, much as a chauffeur controls a car with a steering wheel. Software (including firmware) is layered as shown in Figure 2-2. A user interacts with applications software, which interacts with the OS, which interacts with BIOS and device drivers, which interact with the hardware.

In this chapter, you will learn many of the details of the boot process, which is partly performed by startup BIOS, playing the role of a mechanic who prepares the car for a trip by thoroughly checking critical systems. When the car is in acceptable condition, the mechanic turns the car over to a chauffeur who does some additional minor checking of resources, such as determining the amount of gas in the car and whether the brakes work. This initial startup process performed by the chauffeur compares to startup routines performed by the OS after control is given to it by startup BIOS. Now all is ready to begin. The combination of software programs—applications software, the OS, the BIOS, and device drivers—play the role of a chauffeur, who in addition to keeping an eye on the status of the car, knows how to operate and direct the vehicle.

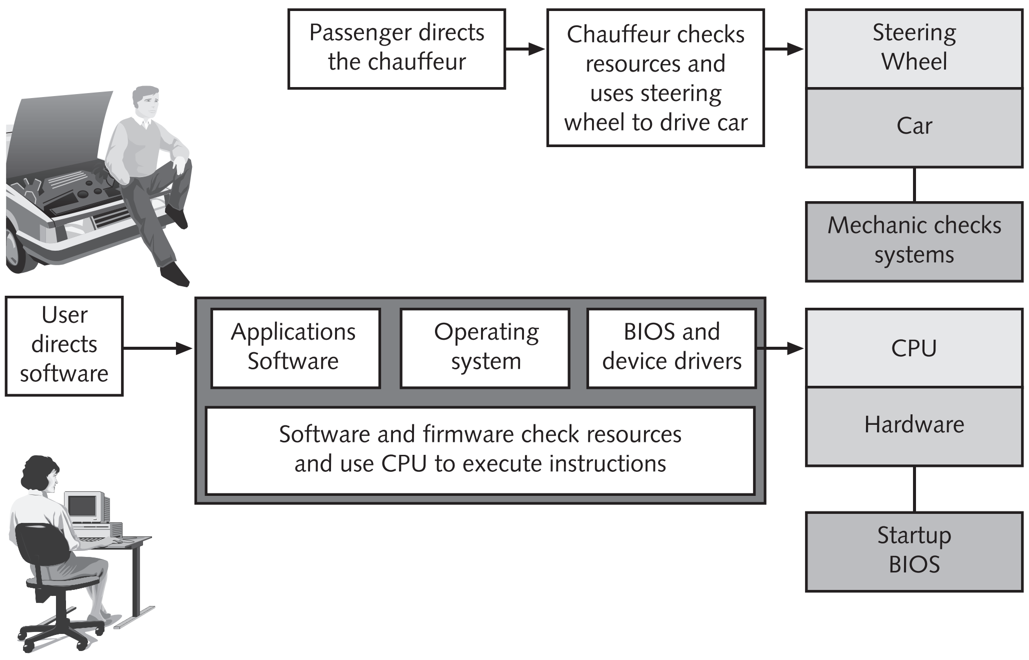

Passenger directs the chauffeur
Chauffeur checks resources and uses steering wheel to drive car
Steering Wheel
Car
Mechanic checks systems
User directs software
Applications Software
Operating system
BIOS and device drivers
Software and firmware check resources and use CPU to execute instructions
CPU
Hardware
Startup BIOS

Figure 2-1 A user interacts with a computer much as a passenger interacts with a chauffeured car

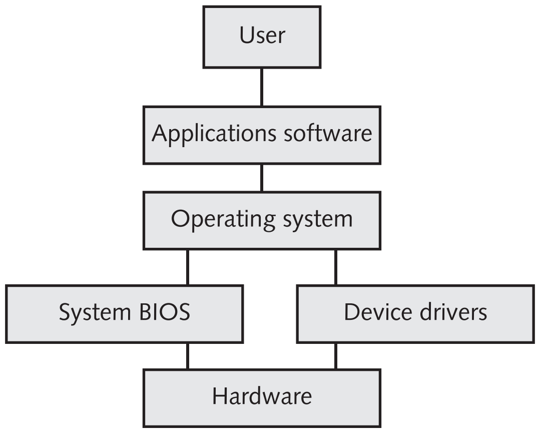


Figure 2-2 The user interacts with applications software that interfaces with the OS; the OS uses either BIOS or device drivers to interface with the hardware

But, remember that the chauffeur will only go where his passenger—who is analogous to a computer user—tells him to go. Thus, the various players work together: the mechanic (BIOS startup program) checks the car (hardware) to make sure it is ready to be used; the passenger (user) provides specific instructions to the chauffeur (software) on what to do or where to go; and the chauffeur (software) interacts directly with the steering wheel (CPU) to control the various underlying mechanisms that make the car (computer) work as intended.

Just as a chauffeur has resources or tools to control a car (for example, controls on the console), and the car has resources to alert the chauffeur that it needs attention (for example, the oil warning light), software has resources to control hardware, and hardware has resources to alert software that it needs attention. Think of a system resource as a tool used by either hardware or software to communicate with the other.

System Resources Defined

There are four types of system resources: memory addresses, I/O addresses, interrupt request numbers (IRQs), and direct memory access (DMA) channels. Table 2-1 lists these system resources used by software and hardware and defines each.

Table 2-1 System resources used by software and hardware

System Resource	Purpose
IRQ	A line of a system-board bus that a hardware device can use to signal the CPU that the device needs attention. Some lines have a higher priority for attention than others. Each IRQ line is identified by a single number.
I/O addresses	Numbers assigned to hardware devices that software uses to get devices' attention and to interact with them. Each device "listens" for these numbers and responds to the ones assigned to it.
Memory addresses	Numbers that are assigned to physical memory located either in RAM or ROM chips. Software can then access this memory by using these addresses.
DMA channel	A number designating a channel whereby the device can pass data to memory without involving the CPU. Think of a DMA channel as a shortcut for data moving to/from the device and memory.

As you can see from Table 2-1, all four resources are used for communication between hardware and software. Hardware devices signal the CPU for attention using an IRQ. Software addresses a device by one of its I/O addresses. Software looks at memory as a hardware device and addresses it with memory addresses, and DMA channels are used to pass data back and forth between a hardware device and memory.

The Boot Process, Step by Step

The processes that occur when a computer is booted are vital to ensuring that it will operate as desired. The functions performed during the boot are:

- Startup BIOS tests essential hardware components. This test is called the **power-on self test** (**POST**).
- Setup information is used to configure both hardware and software.

- Hardware components are assigned system resources that they will later use for communication.
- The OS is loaded, configured, and executed.
- Hardware devices are matched up with the BIOS and device drivers that control them.
- Some applications software may be loaded and executed.

Booting comes from the phrase "lifting yourself up by your bootstraps," and refers to the computer bringing itself up to an operable state without user intervention. Booting refers to either a "soft boot" or "hard boot." A **hard boot**, or **cold boot**, involves initially turning on the power with the on/off switch. A **soft boot** or **warm boot** involves using the operating system to reboot. For DOS, pressing three keys at the same time—Ctrl, Alt, and Del—performs a soft boot. For Windows 95 or 98 (referred to here as Windows 9x), one way to soft boot is to click Start, click Shut Down, and then click OK.

A hard boot is more stressful on your machine than a soft boot because of the initial power surge through the equipment. Always use the soft boot method to restart unless the soft boot method doesn't work. If you must power down, avoid turning off the power switch and immediately turning it back on without a pause, because this can damage the machine. Most PCs have a reset button on the front of the case. Pressing the reset button starts the booting process at an earlier point than does the operating-system method and is, therefore, a little slower, but might work when the operating-system method fails. For newer system boards, pressing the reset button is the same thing as powering off and on except that there is no stress to the system caused by the initial power surge.

The most popular desktop operating system today is Windows 9x, but, because Windows 9x has its roots in DOS, we first cover the details of how DOS is booted and then explain the differences between booting it and Windows 9x.

A+ OS 2.3

The boot process can be divided into four parts: POST, loading the OS, the OS initializing itself, and finally loading and executing an application. Here is a brief overview of all four before we look at each one in detail.

Step 1: POST. The ROM BIOS startup program surveys hardware resources and needs, and assigns system resources to meet those needs (see Figure 2-3). The ROM BIOS startup program begins the startup process by reading configuration information stored in DIP switches, jumpers, and the CMOS chip, and then comparing that information to the hardware—the CPU, video card, disk drive, hard drive, and so on. Some hardware devices have BIOSs of their own that request resources from startup BIOS, which attempts to assign these system resources as needed.

A+ OS 2.3

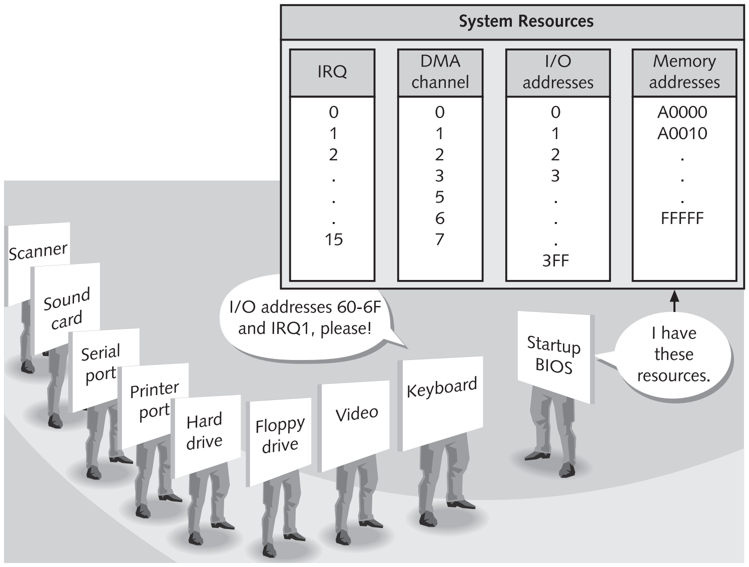


Figure 2-3 Boot Step 1: ROM BIOS startup program surveys hardware resources and needs and assigns system resources to satisfy those needs

Step 2: The ROM BIOS startup program searches for and loads an OS. Most often the OS is loaded from logical drive C on the hard drive (Figure 2-4). Configuration information on the CMOS chip tells startup BIOS where to look for the OS (Figure 2-5). The BIOS turns to that device, reads the beginning files of the OS, copies them into memory, and then turns control over to the OS.

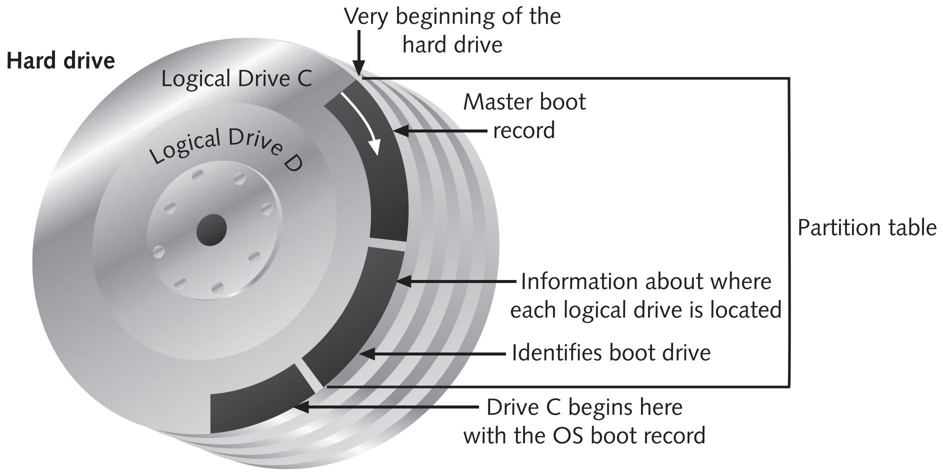

Hard drive
Logical Drive C
Logical Drive D
Very beginning of the hard drive
Master boot record
Partition table
Information about where each logical drive is located
Identifies boot drive
Drive C begins here with the OS boot record

Figure 2-4 A hard drive might contain more than one logical drive; the partition table at the beginning of the drive contains information about the location of each logical drive, indicates which drive is the boot drive, and holds the master boot record that begins the boot process for the operating system

A+ OS 2.3

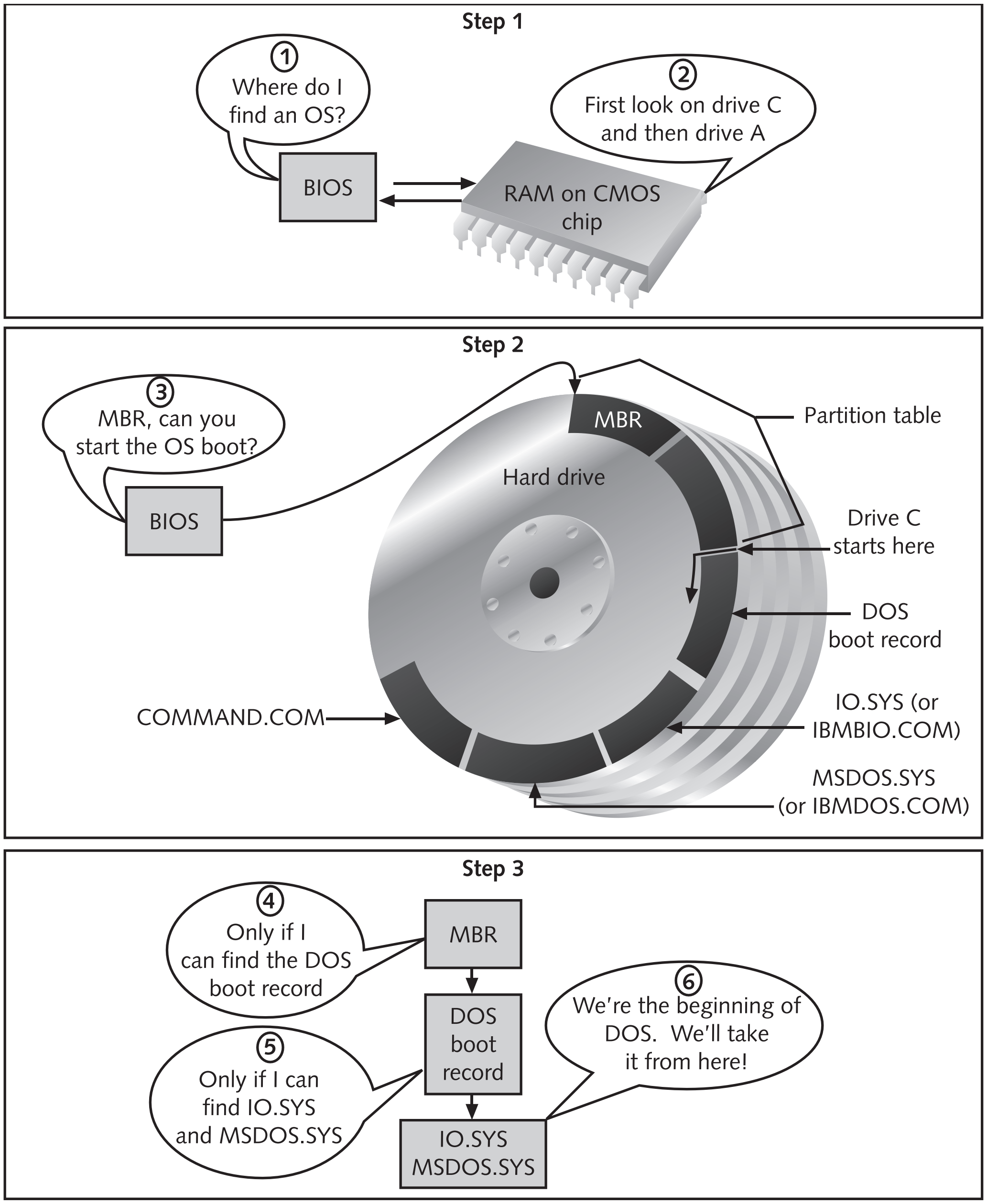

Step 1
1 Where do I find an OS?
2 First look on drive C and then drive A
BIOS
RAM on CMOS chip
Step 2
3 MBR, can you start the OS boot?
BIOS
MBR
Hard drive
Partition table
Drive C starts here
DOS boot record
IO.SYS (or IBMBIO.COM)
MSDOS.SYS (or IBMDOS.COM)
COMMAND.COM
Step 3
4 Only if I can find the DOS boot record
MBR
DOS boot record
IO.SYS MSDOS.SYS
5 Only if I can find IO.SYS and MSDOS.SYS
6 We're the beginning of DOS. We'll take it from here!

Figure 2-5 Boot Step 2: BIOS searches for and begins to load an operating system (in this example, DOS is the OS)

Step 3: The OS configures the system and completes its own loading (see Figure 2-6). The OS checks some of the same things that startup BIOS checked, such as available memory and whether that memory is reliable. Additionally, the OS continues beyond that by loading the software to control a mouse, CD-ROM, scanner, and other

A+ OS 2.3

peripheral devices. These devices generally have programs to manage them, called device drivers, stored on the hard drive. Figure 2-6 shows how DOS core components and applications are loaded. Details of this loading process are covered later in the chapter.

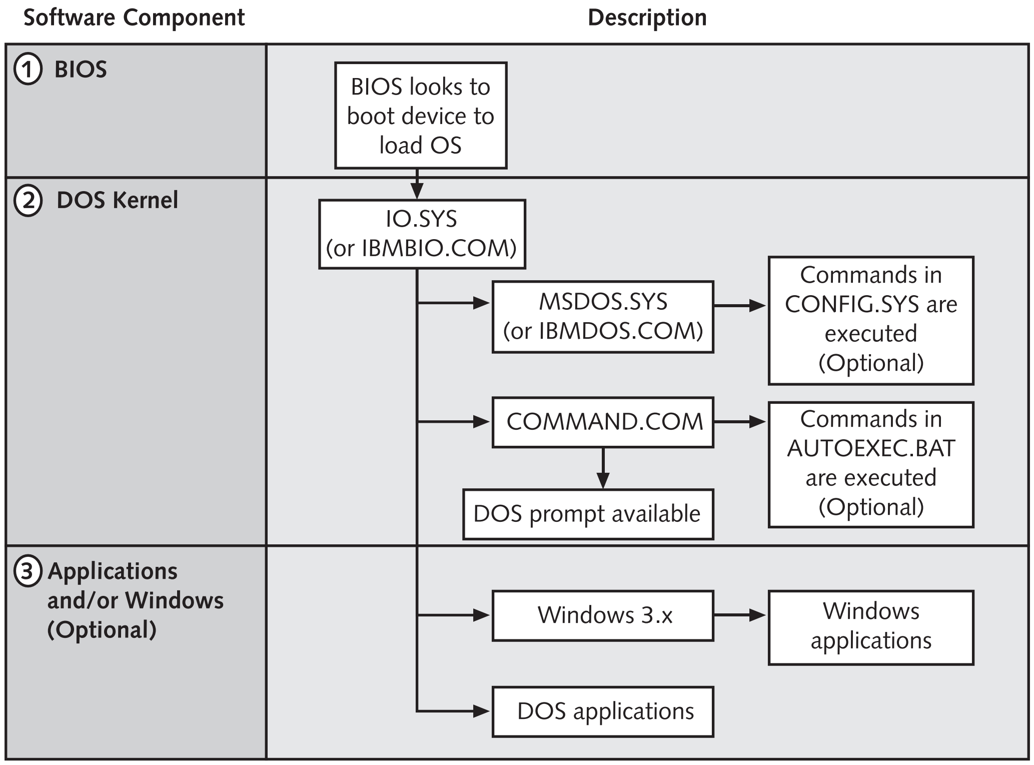

Software Component
Description
1 BIOS
BIOS looks to boot device to load OS
2 DOS Kernel
IO.SYS (or IBMBIO.COM)
MSDOS.SYS (or IBMDOS.COM)
Commands in CONFIG.SYS are executed (Optional)
COMMAND.COM
Commands in AUTOEXEC.BAT are executed (Optional)
DOS prompt available
3 Applications and/or Windows (Optional)
Windows 3.x
Windows applications
DOS applications

Figure 2-6 Boot Step 3: Operating system completes the boot process; DOS core components and applications are loaded

Step 4: The user executes applications software. When you tell the OS to execute an application, the OS first must find the applications software on the hard drive, CD-ROM, or other secondary storage device, copy the software into memory, and then turn control over to it. Finally, you can command the applications software, which makes requests to the OS, which, in turn, uses the system resources, system BIOS, and device drivers to interface with and control the hardware. Your trip has begun!

Step 1: Power-on Self Test (POST)

When you turn on the power to a PC, the CPU begins the process by initializing itself and then turning to the ROM BIOS for instructions. The ROM BIOS then performs POST. Listed below are the key steps in this process.

- When the power is first turned on, the system clock begins to generate clock pulses.

- The CPU begins working and initializes itself (resetting its internal values).
- The CPU turns to memory address FFFF0h, which is the memory address always assigned to the first instruction in the ROM BIOS startup program.
- This instruction directs the CPU to run the POST tests.
- POST first checks the BIOS program operating it and then tests CMOS RAM.
- A test determines that there has not been a battery failure.
- Hardware interrupts are disabled (this means that pressing a key on the keyboard or using another input device at this point will not affect anything).
- Tests are run on the CPU and it is further initialized.
- A check determines if this is a cold boot. If so, the first 16 KB of RAM is tested.
- Hardware devices installed on the computer are inventoried and compared to configuration information.
- Video, memory, keyboard, floppy disk drives, hard drives, ports, and other hardware devices are tested and configured, and IRQ, I/O addresses, and DMA assignments made. The OS will later complete this process.
- Some devices are set up to go into "sleep mode" to conserve electricity.
- The DMA controller is checked.
- Interrupt vectors are moved into the interrupt vector table (this table is covered later in this chapter).
- The interrupt controller is checked (its purpose is explained later in this chapter).
- CMOS setup (a BIOS program to change CMOS configuration data) is run if requested.
- BIOS begins its search for an OS.

A+CORE 2.1

During POST, before the CPU has checked the video system, errors encountered up to this point are communicated by beeps. Short and long beeps indicate an error. Appendix A lists some of these error codes and their meanings. After POST checks and verifies the video controller card (note that POST does not check to see if a monitor is present or working), POST can use the monitor to display its progress. After checking video, POST checks RAM by writing and reading data. A running count of RAM is displayed on the monitor during this phase.

Next, the keyboard is checked, and if you press and hold any keys at this point, an error occurs. Secondary storage, including floppy disk drives and hard drives, is checked. The hardware that POST finds is checked against the data stored in the CMOS chip, jumpers, and/or DIP switches to determine if they agree.

System resources are assigned to devices by more than one method. Jumpers and DIP switches might be set to request a resource (for example, a jumper might be set "on" if IRQ 5 is requested or "off" if IRQ 7 is requested), or the resources needed might simply be "hard coded" into the BIOS. "Hard coded" means that the values of resources are part of the ROM programming and cannot be changed.

For earlier computers, system resources were always assigned to the device during the booting process (see Figure 2-3). Think of the process as a dialog: The startup BIOS recognizes that a hardware device is present. The BIOS asks the device, "What resources do you need?" The device says, "I need this IRQ, these I/O addresses, this DMA channel, and these addresses in upper memory for my BIOS." In almost all cases, a device is the sole owner of these resources. Problems occur when more than one device attempts to use the same resource.

Today, more cooperative Plug and Play devices (introduced in Chapter 1) simply say, "I need one IRQ, some I/O addresses, and this many memory addresses for my BIOS. Please tell me the resources I can use." Plug and Play is discussed in detail in Chapter 12.

Step 2: BIOS Finds and Loads the OS

Once POST is complete, the next step is to load an OS. Most often the OS is loaded from the hard drive (see Figure 2-4). The minimum information required on the hard drive to load an OS is:

A+ OS 1.1

- A small program at the very beginning of the hard drive called the master boot program that is needed to locate the beginning of the OS on the drive.
- A table that contains a map to the logical drives on the hard drive, including which drive is the boot drive. This table is called the partition table.
- At the beginning of the boot drive (usually drive C) the OS boot record that loads the first program file of the OS. For DOS, that program is IO.SYS.
- For DOS, MSDOS.SYS is needed next, followed by COMMAND.COM. These two files plus IO.SYS are the core components of DOS.

Often a hard drive is divided or partitioned into more than one logical drive—for example, drive C and drive D, as seen in Figure 2-4. Whether a hard drive has one or several logical drives, it always contains a single partition table located at the very beginning of the drive. At the beginning of the table is a small program used to start the boot process from the hard drive, which is called the master boot program, or master boot record (MBR). One logical drive on the drive is designated as the boot drive, and the OS is stored on it. At the beginning of this logical drive is the OS boot record that knows the names of the files that contain the core programs of the OS.

The process for BIOS to load the OS begins with BIOS looking to CMOS setup to find out which secondary storage device should have the OS (see Figure 2-5). Setup might instruct the BIOS to first look to drive C, and, if no OS is found there, then try drive A; or the order might be A then C. If BIOS looks first to drive A and does not find a disk in the drive, it turns to drive C. If it first looks to drive A and finds a disk in the drive, but the disk does not contain the OS (for DOS, that means the DOS boot record, IO.SYS, MSDOS.SYS, and COMMAND.COM), then this error message is displayed:

```
Non-system disk or disk error, replace and press any key
```

You must replace the disk with one that contains the OS or simply remove the disk to force the BIOS to continue on to drive C to find the OS. In the next section, we examine the details of booting using MS-DOS, Windows 95, and Windows 98. A discussion of booting using Windows NT is left to Chapter 13, and booting Windows 2000 is covered in Chapter 14.

Step 3: The OS Completes the Boot Process

This section describes what first happens during booting when DOS is loaded as the OS. In Step 2 of Figure 2-5, the BIOS locates the master boot record on the hard drive, which looks to the partition table to determine where the logical boot drive is physically located on the drive. It then turns to the DOS boot record of that logical drive.

The DOS boot record is a very short program; it loads just two hidden files which make up the core part of DOS (sometimes called the DOS kernel), into memory (see Figure 2-5, Step 3 and Figure 2-6). (A **hidden file** is a file not displayed in the directory list.) The DOS boot record program knows the filenames, which are IO.SYS and MSDOS.SYS for Microsoft DOS, and IBMBIO.COM and IBMDOS.COM for IBM DOS. You'll see IBM DOS on IBM and Compaq PCs and Microsoft DOS on other PCs. The IO.SYS file contains more BIOS software. MSDOS.SYS contains software to manage files, run applications software, and interface with hardware. Once these two files are loaded into memory, the boot record program is no longer needed, and control turns to a program stored in MSDOS.SYS. This program looks on the hard drive for a file named CONFIG.SYS. CONFIG.SYS is the first OS file that you, as a user, can change. This configuration file contains commands that tell DOS how many files it can open at any one time (FILES=) and how many file buffers to create (BUFFERS=). (A buffer is a temporary holding area for files.) CONFIG.SYS also includes the commands to load device drivers (DEVICE=) as well as other information. (Remember a driver is a program that instructs the computer how to communicate with and manage a peripheral device such as a modem or scanner.) An example of a typical command in CONFIG.SYS is the following:

```
DEVICE=C:\SCANGAL\SCANNER.SYS
```

This command line tells DOS to look in a directory named \SCANGAL on drive C for a file named SCANNER.SYS, copy it into memory, and save it there until an application requests to use a scanner. The SCANNER.SYS program tells DOS how to communicate with that scanner. Several drivers can be loaded into memory from commands in CONFIG.SYS. DOS puts these programs in memory wherever it chooses. However, a program can request that it be put in a certain memory location.

After CONFIG.SYS is executed, MSDOS.SYS looks for another DOS file named COMMAND.COM. This file has three parts: more code to manage I/O, programs for internal DOS commands such as COPY and DIR, and a short program that looks for another file named AUTOEXEC.BAT.

The filename **AUTOEXEC.BAT** stands for **automatically executed batch** program. The file lists DOS commands that are executed automatically each time DOS is loaded. The following two commands (or similar prompt and path commands) are typically in the AUTOEXEC.BAT file:

```
PROMPT $P$G
PATH C:\;C:\WINDOWS;C:\WP51;C:\DBASE;C:\123
```

The PROMPT command tells DOS to display the current directory name and the current drive name as part of the prompt. Without the prompt command, your prompt looks like this: C>. With the prompt command it might look like C:\WP51> where the WP51 tells you which directory is current.

Recall that you learned about the PATH command in Chapter 1. The PATH command shown above lists five paths separated by semicolons. This command directs DOS to look in five different directories for program files. Recall that without the benefit of the PATH command, you execute application programs in one of two ways: go to the directory containing the application by using the CD (change directory) command, or include the path with the name of the executable program file when you execute the software. Sometimes during the installation of a software package, the installation process automatically adds a new path to the existing PATH command in AUTOEXEC.BAT, telling DOS where the new application can be found.

Rather than appending an entry to the existing PATH command line, some software installation programs use the SET command to append a path to the PATH command without editing the existing PATH command line itself. It works like this:

```
SET PATH=%PATH%;C:\VERT
```

The path C:\VERT is appended to the existing PATH command.

The SET command is also used to create and assign a value to an environmental variable that can later be read by an application. A software installation program might add a SET command to your AUTOEXEC.BAT file. This command might look something like this:

```
SET MYPATH=C:\VERT
```

Later the software will use the environmental variable MYPATH in the program.

TIP Sometimes an installation program places the SET command too late in the AUTOEXEC.BAT file (after the WIN command to load Windows). If the application runs under Windows, Windows loads before the variable is set, which can cause an application error. The solution is to move the SET command before the WIN command in the AUTOEXEC.BAT file. Reboot the PC for the change to take effect.

Another typical use of AUTOEXEC.BAT is to load TSRs. A **TSR** is a "terminate-and-stay-resident" program that is loaded into memory but not immediately executed. The program later executes when some "hot" key is pressed or a special hardware action occurs, such as moving the mouse. An example of a TSR is a screen capture program. The program is loaded by entering the following command in AUTOEXEC.BAT:

```
C:\CAPTURE\SAVEIT.EXE
```

In this example, SAVEIT.EXE is the name of the program in the directory \CAPTURE. For this software, the hot key to activate the program is the Print Screen key. Later, when you are using any applications software and you press the Print Screen key, instead of the screen actually printing under DOS as it normally would, the SAVEIT.EXE program saves a copy of the screen to a graphics file or offers you other options. When you exit the screen capture program, control returns to the application, where you continue working until you press Print Screen to evoke the TSR again.

Completion of the Boot Process

The boot process is completed after AUTOEXEC.BAT has finished executing. At this point, COMMAND.COM is the program in charge, providing you with a command prompt and waiting for your command. On the other hand, if a program was executed from AUTOEXEC.BAT, it might ask you for a command.

In a Windows 3.x environment, it is common to include in AUTOEXEC.BAT the following command to execute Windows each time the computer is booted. (The file extension is usually omitted, but can be included.)

```
C:\WINDOWS\WIN
```

The file that is executed above is WIN.COM found in the C:\WINDOWS directory. Windows software loads just like other software running under DOS. The program is first loaded into memory and then executed. WIN.COM then looks for several configuration files that contain user and environmental settings, such as the size of the screen display font, the desktop colors used by the monitor, and the speed of the mouse. These configuration files, sometimes called initialization files, or .ini files, are listed in Table 2-2 and are usually stored in the same directory as WIN.COM.

Table 2-2 Windows configuration files

Windows Configuration Files	General Purpose of the File
SYSTEM.INI	Contains hardware settings and multitasking options for Windows
PROGMAN.INI	Contains information about Program Manager groups
WIN.INI	Contains information about user settings, including printer, fonts, file associations, and settings made by applications
CONTROL.INI	Contains information about the user's desktop, including color selections, wallpaper, and screen saver options
MOUSE.INI	Contains settings for the mouse

Editing AUTOEXEC.BAT and CONFIG.SYS

When using DOS, you can change CONFIG.SYS and AUTOEXEC.BAT to configure your OS environment, install TSRs, or troubleshoot boot problems. You can change the contents of CONFIG.SYS and AUTOEXEC.BAT with any text editor. DOS provides EDIT, a full-screen text editor.

There is a risk in changing AUTOEXEC.BAT. If you make a mistake, your computer can stall during the boot process, making it impossible to use without rebooting. Because of this risk, never change AUTOEXEC.BAT without first making a bootable disk that you can use if your AUTOEXEC.BAT file on the hard drive fails or causes problems.

To make a bootable disk and a backup copy of AUTOEXEC.BAT:

1. From the DOS prompt, type:

   ```
   C:\> FORMAT A:/S
   ```

The command erases any files currently on the disk in drive A and the /S switch copies the two DOS hidden files and COMMAND.COM to the disk in drive A, making the disk bootable.

2. If your computer shows either drive A or B as the default drive, make drive C the default, as follows:

   ```
   A:\> C:
   ```

3. If the default directory is not the root directory, make it so, as follows:

   ```
   C:\WINDOWS> CD\
   C:\>
   ```

   No matter what the default directory was (in this example, it was \WINDOWS), the backslash (\) in the prompt indicates that the root is now the default.

4. Back up the current copy of AUTOEXEC.BAT to the hard disk, as follows:

   ```
   C:\> COPY AUTOEXEC.BAT AUTOEXEC.BK
   ```

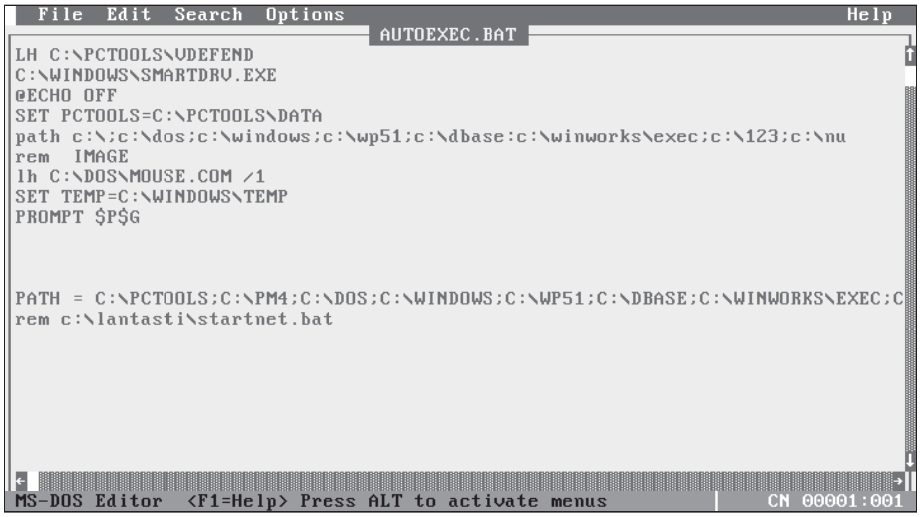
   or to a disk in drive A, as follows:

   ```
   C:\> COPY AUTOEXEC.BAT A:
   ```

5. Edit the file on drive C, as follows:

   ```
   C:\> EDIT AUTOEXEC.BAT
   ```

Your screen should be similar to that shown in Figure 2-7. Follow the directions on the screen to use and exit the editor.



```
LH C:\PCTOOLS\VDEFEND
C:\WINDOWS\SMARTDRV.EXE
@ECHO OFF
SET PCTOOLS=C:\PCTOOLS\DATA
path c:\;c:\dos;c:\windows;c:\wp51;c:\dbase:c:\winworks\exec;c:\123;c:\nu
rem  IMAGE
lh C:\DOS\MOUSE.COM /1
SET TEMP=C:\WINDOWS\TEMP
PROMPT $P$G


PATH = C:\PCTOOLS;C:\PM4;C:\DOS;C:\WINDOWS;C:\WP51;C:\DBASE;C:\WINWORKS\EXEC;C
rem c:\lantasti\startnet.bat
```

```
MS-DOS Editor  <F1=Help> Press ALT to activate menus       CN 00001:001
```

Figure 2-7 Edit AUTOEXEC.BAT

You must remove the disk from drive A and reboot your computer (using Ctrl+Alt+Del) to execute the new AUTOEXEC.BAT file on your hard drive. If the computer stalls during the boot, place the bootable disk in drive A and then reboot. Remember that if the disk does not have a copy of AUTOEXEC.BAT on it, you do not have an active PATH command.

Do not use word-processing software, such as Word or WordPerfect, to edit AUTOEXEC.BAT, unless you use the ASCII text mode, because word-processing applications place control characters in their document files that prevent DOS from interpreting the AUTOEXEC.BAT file correctly.

Today, most computers running DOS allow you to step through each line of the CONFIG.SYS and AUTOEXEC.BAT files if you press the F8 function key as soon as the "Starting MS-DOS" message appears during the boot process. This feature can be a very helpful debugging tool.

A+ OS 1.1

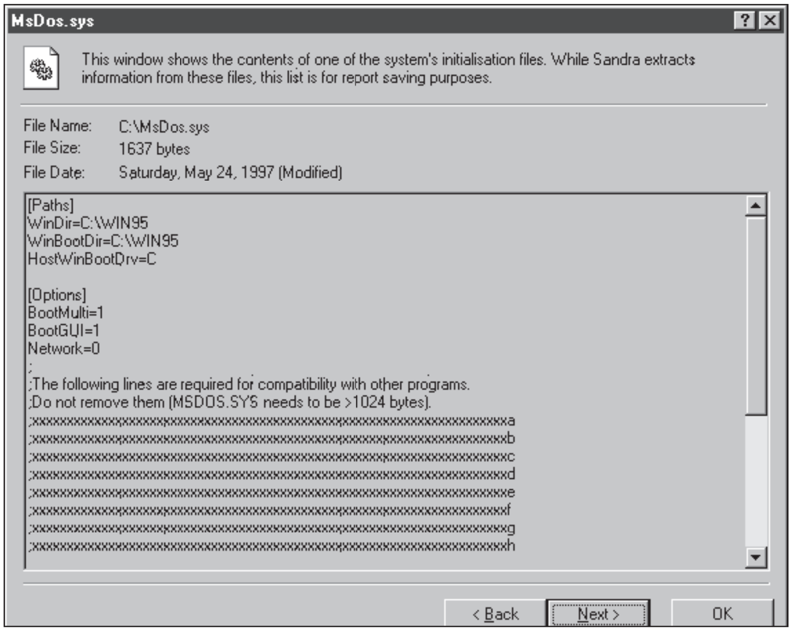
Booting with Windows 95

Recall that if a computer takes full advantage of the Windows 95 OS, its BIOS is Plug and Play BIOS, meaning that BIOS configures the Plug and Play devices before it loads Windows 95. A Plug and Play device allows BIOS to select the device's computer resources, such as an IRQ, an I/O address, and DMA channels. BIOS then turns this information over to Windows 95 when it loads. The process is described below:

1. When you boot the machine, POST occurs just as it does for BIOS that is not Plug and Play.
2. The Plug and Play BIOS begins by examining the devices on the system and determining which ones are Plug and Play compliant. BIOS first enables the devices that are not Plug and Play, and then tries to make the Plug and Play devices use the leftover resources.
3. BIOS looks for a device containing the OS and loads Windows 95, making information about the current allocation of resources available to the OS.
4. Just as with DOS, the master boot record executes the boot record on the hard drive, which looks for the initial hidden file of Windows 95, called IO.SYS.
5. Again, just as with DOS, IO.SYS loads. In fact, IO.SYS is really a small core DOS module. IO.SYS looks for a CONFIG.SYS file, and, if found, the CONFIG.SYS file executes. Windows 95 does not require CONFIG.SYS because many of its functions are incorporated into Windows 95, but you can use CONFIG.SYS to load a device driver if you choose. However, CONFIG.SYS loads only 16-bit drivers which are slower than the 32-bit drivers that Windows 95 loads.
6. After CONFIG.SYS is complete, IO.SYS looks for MSDOS.SYS. The role of MSDOS.SYS in Windows 95 is much different from its role in DOS. In Windows 95, MSDOS.SYS is a hidden text file containing settings to customize the boot process. It follows a format similar to that of the .ini files of Windows 3.x.

A+ OS 1.1

Typical lines at the beginning of MSDOS.SYS look like those shown in Figure 2-8. The functions of the first few MSDOS.SYS entries are listed in Table 2-3.



This window shows the contents of one of the system's initialisation files. While Sandra extracts information from these files, this list is for report saving purposes.

File Name: C:\MsDos.sys
File Size: 1637 bytes
File Date: Saturday, May 24, 1997 (Modified)

```
[Paths]
WinDir=C:\WIN95
WinBootDir=C:\WIN95
HostWinBootDrv=C

[Options]
BootMulti=1
BootGUI=1
Network=0
;
;The following lines are required for compatibility with other programs.
;Do not remove them (MSDOS.SYS needs to be >1024 bytes).
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxa
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxb
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxc
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxd
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxe
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxf
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxg
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxh
```

Figure 2-8 Sample MSDOS.SYS file from Windows 9x

Table 2-3 Entries in the MSDOS.SYS file for Windows 9x

Entry	Description
WinDir=	Location of the Windows 9x directory
WinBootDir=	Location of the Windows 9x startup files
HostWinBootDrv=	Drive that is the Windows boot drive
BootGUI=	When BootGUI=1, automatic graphical startup into Windows 9x is enabled When BootGUI=0, the system boots to a command prompt

7. Next, COMMAND.COM loads just as it does with DOS. COMMAND.COM provides a command interface for users and executes an AUTOEXEC.BAT file if it is present.
8. If AUTOEXEC.BAT is found, it now executes.
9. The heart of Windows 95 now loads, providing a desktop from which you can execute applications software. Further details regarding booting with Windows 95 are covered in Chapter 12.

A+ OS 1.1

You can see from this description of loading Windows 95 that it still includes many DOS functions. In fact, by using the BootGUI entry in MSDOS.SYS, it is possible to backtrack from a Windows 95 installation to the underlying DOS 7.0, which comes with Windows 95.

Booting with Windows 98

Just as with Windows 95, Windows 98 goes through the boot sequence in the order listed above. First BIOS runs POST, then it loads a small DOS core, and then this DOS core loads Windows 98. Windows 98 has made some minor changes in what happens during startup to speed up the boot process. For instance, Windows 95 waits two seconds while "Starting Windows 95" is displayed so that you can press a key to alter the boot process. Windows 98 eliminated this two-second wait and, in its place, allows you to press and hold the Ctrl key as it loads. If you do that, you see the Startup Menu that is also available with Windows 95.

How Software Manages Hardware Resources

Recall from earlier in the chapter that the four system resources are IRQs, I/O addresses, memory addresses, and DMA channels. Their definitions are listed in Table 2-1. All four of these system resources are dependent on certain lines on a bus on the system board. Some lines on the bus are devoted to IRQs, some to addresses (both memory addresses and I/O addresses), and some to DMA channels. It's impossible to truly understand these resources without relating them to physical lines on the bus. Also, relating these logical resources to something physical (and visual) makes the concepts easier to understand. Therefore, we next turn our attention to a careful examination of one system-board bus before we look back at understanding system resources.

The 8-bit and 16-bit ISA Bus

Recall from Chapter 1 that all devices are directly or indirectly connected to the system board because they are all dependent on the CPU for processing their data. A device connects to the system board by a data cable, a slot, or a port coming directly off the system board. In any case, the device always connects to a single bus on the system board. Recall that there are several different buses on a system board, but our discussions here will be limited to only one bus, an older bus used on the early PCs of the 1980s, called the ISA (Industry Standard Architecture) bus, because this simpler bus is easier to understand. The first ISA bus had only eight lines for data and was called the 8-bit ISA bus. Figure 2-9 shows an expansion slot for this bus with some of the pinouts labeled.

A+ CORE 4.3

Some of the lines on the bus are used for data, addresses, and voltage, and others are a variety of control lines. Looking at Figure 2-9, you can see that 8 lines are used for data, and 20 lines are used for addresses. These 20 lines can carry either memory addresses or I/O addresses. The CPU determines which type of address is using these lines by setting control lines B11 through B14, as shown in Figure 2-9 (memory read/write and I/O read/write). Since this bus has only 20 address lines, the largest address value that can travel on the bus is 1111 1111 1111 1111 1111 or 1,048,576 (1024K).

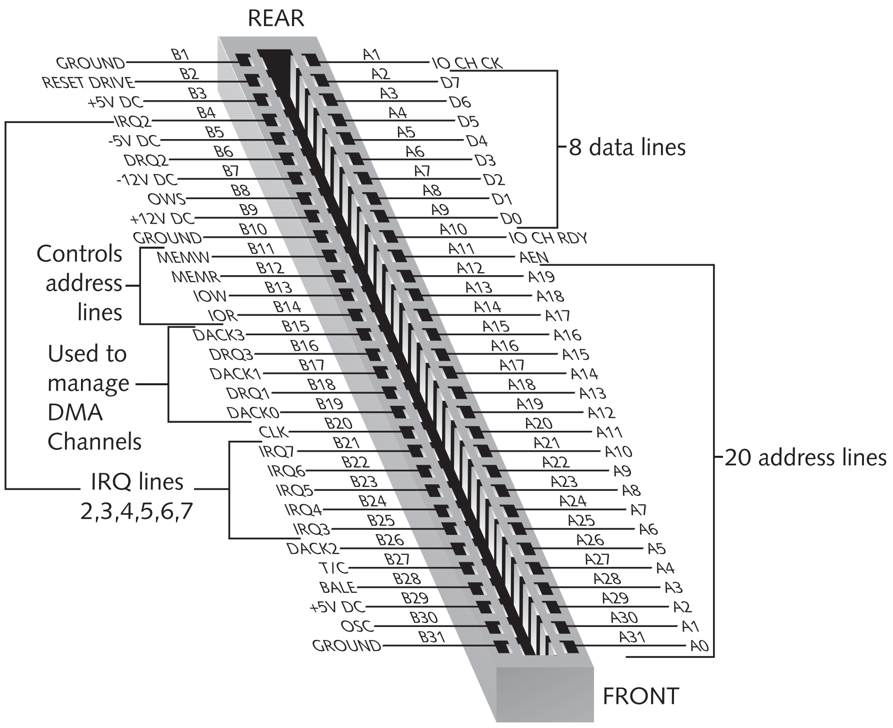


Figure 2-9 A 62-pin expansion slot for the 8-bit ISA bus

Two lines are required to manage a DMA channel: DRQ (Direct Request) and DACK (Direct Acknowledge). There are four DMA channels (0, 1, 2, and 3) on the 8-bit ISA bus.

As computer technology improved, the demand increased for more memory, more devices to be operating at the same time, and faster data transfer, making it necessary to provide more memory addresses, DMA channels, and IRQs. The 16-bit ISA bus was invented to meet these requests. Figure 2-10 shows the 16-bit ISA bus that added an extra extension to the 8-bit slot allowing for 8 additional data lines (total of 16), 5 additional IRQ lines, 4 more DMA channels, and 4 additional address lines (total of 24). Today, the 16-bit ISA bus is still used on system boards, although newer, faster buses are also used. An 8-bit expansion card (an expansion card that only processes 8 bits of data at one time) can use the 16-bit ISA expansion slot. It only uses the first part of the slot.

System boards also contain other newer buses that are faster and provide more options, but the basics haven't changed. You can still find lines on these buses for data, addresses, IRQs, and DMA channels, although it is common practice today for a line to perform several functions, making it not quite as easy (and not as much fun) to study the bus. Now that you've been introduced to what's on a bus, let's turn our attention back to the four system resources, what they are, and how they work.

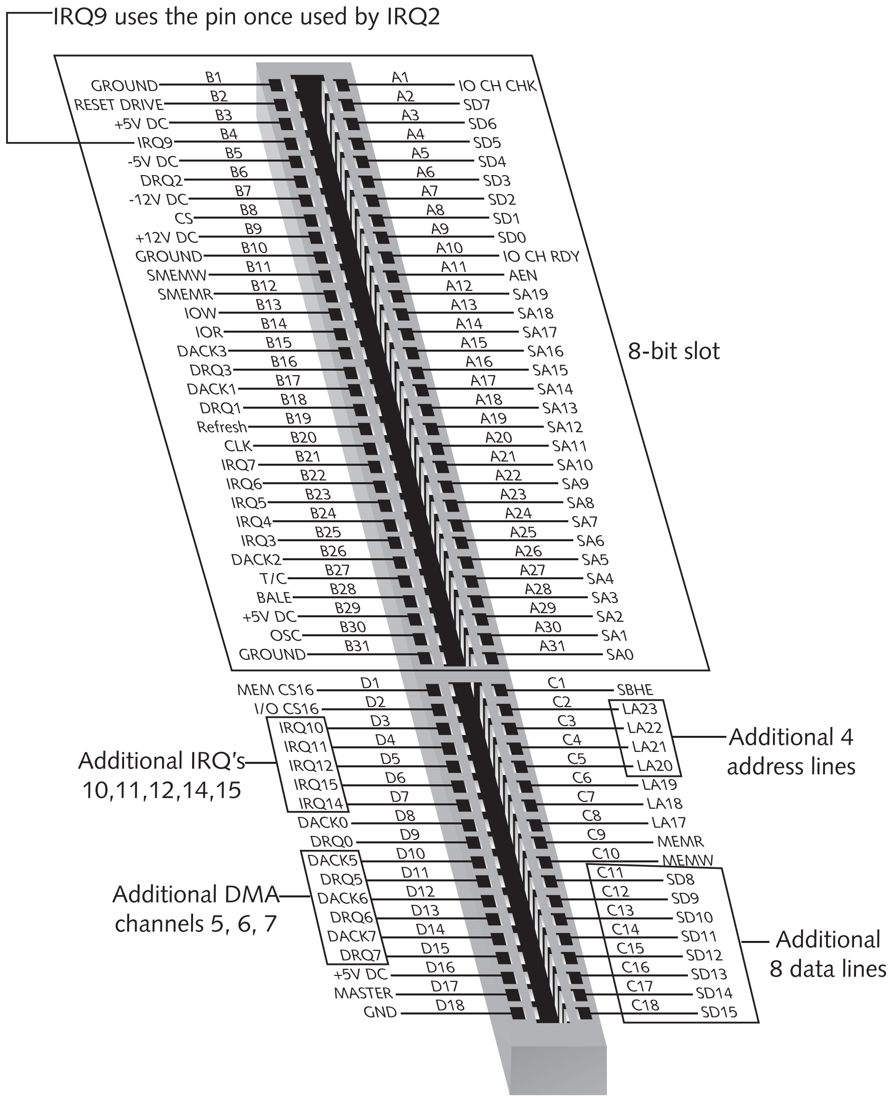


Figure 2-10 A 98-pin expansion slot for the 16-bit ISA bus

Interrupt Request Number (IRQ)

When a hardware device needs the CPU to do something, such as when the keyboard needs the CPU to process a keystroke after a key has been pressed, the device needs a way to get the CPU's attention and the CPU must know what to do once its attraction is turned to the device. These interruptions to the CPU are called **hardware interrupts** and the device handles them by placing voltage on a designated line on the bus it is connected to. These lines are numbered, and a line is referred to as an **interrupt request number**, or **IRQ**. This voltage on the line serves as a signal to the CPU that the device has a request that needs processing. Often, a hardware device that needs attention from the CPU is referred to as "needing servicing."

A+CORE 2.2

Look carefully at Figure 2-9 for the IRQ lines (IRQ 2, 3, 4, 5, 6, and 7). See Table 2-4 for a listing of the common uses for these IRQs. There are actually eight IRQs built into this bus, but IRQs 0 and 1 are not available for expansion cards (because they are used for system timer and keyboard), so they are not given a pin on the expansion slot. IRQ 2 was reserved in the early days of PCs because it was intended to be used as part of a link to mainframe computers. Thus, only five IRQs were available for devices, and each device had to have its own IRQ. This made it difficult for more than five devices to be connected to a PC at any one time. In Table 2-4, notice the COM and LPT assignments. COM1 and COM2 are preconfigured assignments that can be made to serial devices such as modems, and LPT1 and LPT2 are preconfigured assignments that can be made to parallel devices such as printers. You will learn more about these in Chapter 9.

On early system boards, the eight IRQs were managed by an Intel microchip called the interrupt controller chip and labeled the Intel 8259 chip. This chip had a direct connection to the CPU and signaled the CPU when an IRQ was activated. The CPU actually doesn't know which IRQ is "up" because the interrupt controller manages that for the CPU. If more than one IRQ is up at the same time, the interrupt controller selects the IRQ that has the lowest value to process first. For example, if a user presses a key on the keyboard at the exact same time that she moves the mouse installed on COM1, since the keyboard is using IRQ 1 and the mouse on COM1 is using IRQ 4, the keystroke is processed before the mouse action.

Table 2-4 Interrupt Request Numbers for devices using the early 8-bit ISA bus

IRQ	Device	IRQ	Device
0	System timer	4	COM1
1	Keyboard controller	5	LPT2
2	Reserved (not used)	6	Floppy drive controller
3	COM2	7	LPT1

When the 16-bit ISA bus appeared, more IRQs became available. A second interrupt controller chip was added to the system-board chip set, which hooked to the first controller. The second controller used one of the first controller's IRQ values (IRQ 2) to signal the first controller (see Figure 2-11). But there was a problem with that because there were some devices that used IRQ 2. Tying the new IRQ 9 to the old IRQ 2 pin on the 16-bit ISA bus solved the problem. The result was that a device could still use the pin on the expansion slot for IRQ 2, but it is really IRQ 9. (Look carefully at Figure 2-10 to confirm that.) Because of this, the priority level becomes: 0, 1, (8, 9, 10, 11, 12, 13, 14, 15), 3, 4, 5, 6, 7.

A+CORE 2.2

To see how the IRQs are assigned on your computer, use MSD for DOS and Windows 3.x, and Device Manager for Windows 9x. For Windows 9x, click **Start**, **Programs**, **Settings**, **and Control Panel**, and double-click **System**. Click the **Device Manager** tab. Select **Computer** and click **Properties**. Figure 2-12 shows the Computer Properties dialog box. Notice that IRQ 2 is assigned to the programmable interrupt controller, and IRQ 9 is used by the video card.

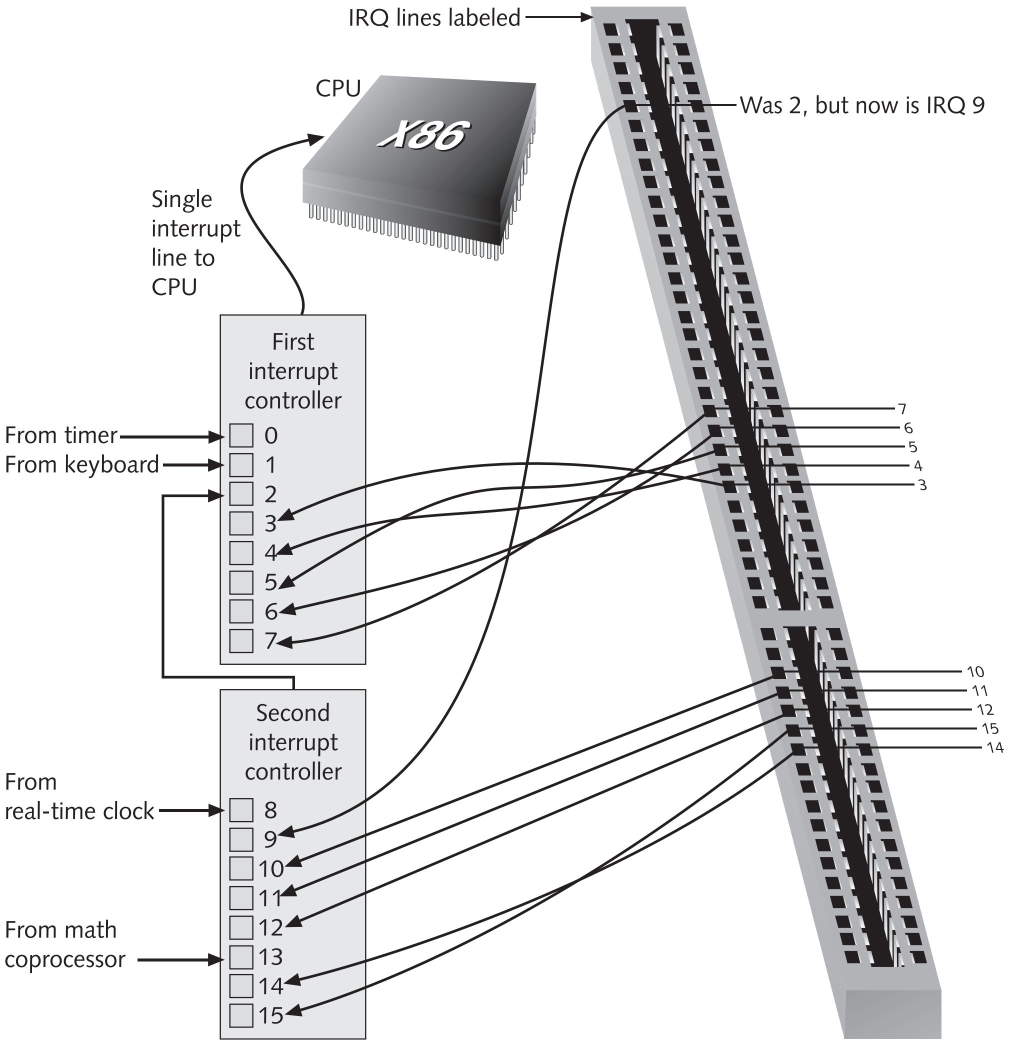
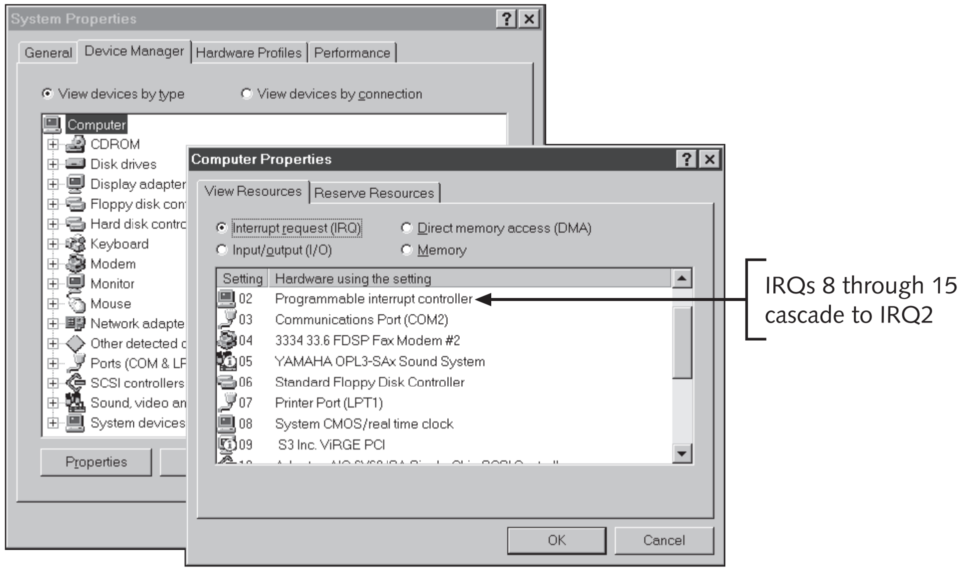


Figure 2-11 When eight more IRQs were added to system boards, IRQ 2 was used to receive all interrupts from these eight. IRQ 9 was wired to the pin on the ISA bus previously assigned to IRQ 2. You can say that IRQs 8–15 "cascade" to IRQ 2.

Newer buses are designed to allow more than one device to share an IRQ. In future chapters you will see how an entire bus only needs one IRQ for all devices and how the bus manages that feat.

Many processes that the CPU carries out are initiated by interrupts and are said to be "interrupt driven." Later in the chapter you will see how software can also issue an interrupt to the CPU so that the software can have access to a device.



[✓ View devices by type] View devices by connection

Computer
CDROM
Disk drives
Display adapter
Floppy disk con
Hard disk contrc
Keyboard
Modem
Monitor
Mouse
Network adapte
Other detected d
Ports (COM & LF
SCSI controllers
Sound, video an
System devices

Computer Properties

View Resources | Reserve Resources

[✓ Interrupt request (IRQ)] Direct memory access (DMA) Input/output (I/O) Memory

Setting	Hardware using the setting
02	Programmable interrupt controller
03	Communications Port (COM2)
04	3334 33.6 FDSP Fax Modem #2
05	YAMAHA OPL3-SAx Sound System
06	Standard Floppy Disk Controller
07	Printer Port (LPT1)
08	System CMOS/real time clock
09	S3 Inc. ViRGE PCI

OK Cancel

Figure 2-12 Use Device Manager to see how IRQs are used by your system

With interrupts, the hardware device or the software initiates communication by sending a signal to the CPU, but a device can be serviced in another way, called polling. With **polling**, software is constantly running that has the CPU periodically check the hardware device to see if service is needed. Not very many devices use polling as the method of communication. A joystick is one example of such a device. Software that is written to manage a joystick has the CPU check the joystick periodically to see if the device has data to communicate, which is why a joystick does not need an IRQ to work. Most hardware devices use interrupts.

Memory Addresses

Once the IRQ gets the attention of the CPU, its job is done, but memory addresses are used as the device is serviced. Recall that memory addresses are numbers assigned to both ROM and RAM memory so that the CPU can access that memory. Think of memory addresses as a single long list of hexadecimal numbers, as described in Appendix D. The CPU has a fixed number of memory addresses available to it as determined by the CPU and the bus it is using. These memory addresses can be assigned to any type of physical memory in the system that needs to be addressed by the CPU, including ROM and RAM chips on expansion cards and ROM and RAM chips on the system board, which can hold either data or instructions. Once addresses have been assigned, the CPU only sees this physical memory as a single list that can be accessed by using the memory addresses.

Also, remember from Chapter 1 that before the CPU can process data or instructions, both must be in physical memory, and the physical memory must be assigned memory addresses. In the case of ROM BIOS, these programs are already located on a memory chip, and so the only thing needed is for memory addresses to be assigned to them. Software stored on a hard drive or other secondary storage device must first be copied into RAM before processing, and memory addresses must be assigned to that RAM. Data coming from a keyboard or some other input device or stored in data files on a hard drive must also be copied into RAM before processing, and memory addresses must be assigned to that RAM. As one example of this process, Figure 2-13 shows that both RAM and memory addresses are needed so the CPU can load a program stored on the hard drive into memory before executing it.

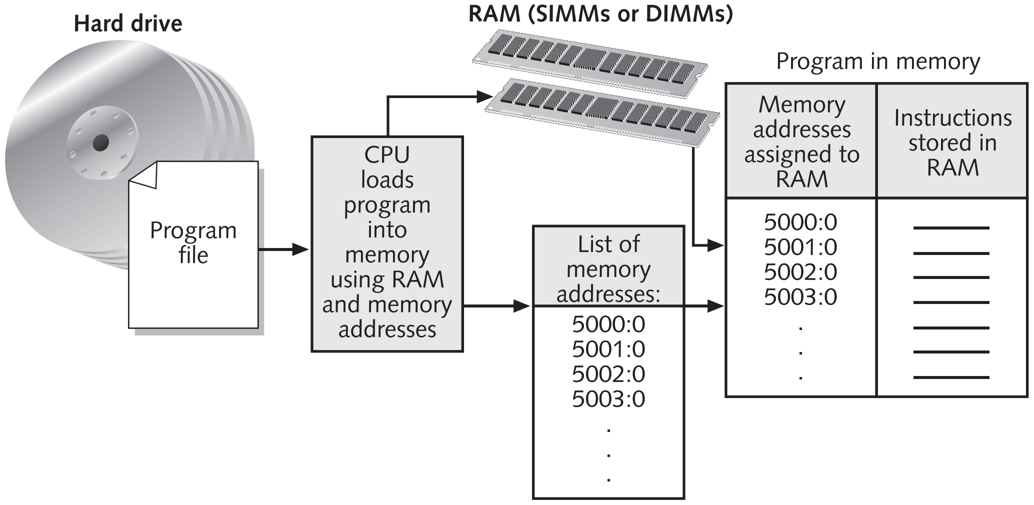

Hard drive
RAM (SIMMs or DIMMs)
Program in memory
Program file
CPU loads program into memory using RAM and memory addresses
List of memory addresses:
5000:0
5001:0
5002:0
5003:0
.
.
.
Memory addresses assigned to RAM
Instructions stored in RAM
5000:0
5001:0
5002:0
5003:0
.
.
.

Figure 2-13 RAM is a hardware resource, and memory addresses are a system resource; both are used when loading software into memory for execution

Table 2-5 Some ways memory addresses are used

Physical Location of Memory	Contents (Data and/or Instructions)
ROM BIOS chip on system board	• Startup BIOS program • System BIOS programs
RAM stored on SIMMs and DIMMs on system board	• Parts of the OS permanently stored on the hard drive • Applications software permanently stored on the hard drive • Device drivers permanently stored on the hard drive • Data used by the OS, device drivers, and applications software, either coming from input devices or copied from secondary storage devices
Video RAM stored on memory chips on a video card	• Video data
Video ROM BIOS chip on a video card	• Programs to control video (video BIOS)
RAM and ROM chips on other expansion cards	• BIOS and data used by the peripheral devices that the card supports (examples are network card, sound card)

How Are Memory Addresses Used?

Table 2-5 shows the ways that the CPU uses its list of memory addresses. Figure 2-14 shows a possible map that could result when all these items in Table 2-5 have claimed their memory addresses. Startup BIOS and **system BIOS**—also called **on-board BIOS**—stored in the ROM-BIOS chip on the system board must be assigned memory addresses so that the CPU can access these programs. RAM stored in SIMMs and DIMMs on the system board makes up the bulk of memory that is used by the CPU and uses the lion's share of available memory addresses. Many programs and data are copied into this RAM, including device drivers, portions of the OS, and applications software and data.

Notice in Figure 2-14 that some device drivers and BIOS are labeled 16-bit and some are labeled 32-bit. Using Windows 9x and DOS, all 16-bit programs including device drivers and BIOS must be assigned memory addresses in the first 1024K of addresses. Faster 32-bit programs can be assigned addresses in extended memory. DOS cannot support 32-bit programs without the help of Windows 3.x, so a pure DOS-based system can only use 16-bit programs. One goal of Windows 9x is to replace all older 16-bit device drivers with newer 32-bit versions. Also, newer devices contain BIOS that is written using 32-bit code rather than 16-bit code. Therefore, in a Windows 9x environment, most of the memory addresses below 1024K are not used.

Finally, note that Figure 2-14 applies to DOS and Windows 9x only. Windows NT and Windows 2000 use an altogether different memory-mapping design where there is no conventional, upper, or extended memory—it's all just memory. With this new approach to memory management, BIOS and device drivers have no say as to what memory addresses they are assigned.

Figure 2-14 shows that some memory addresses are used for video; every computer system has video. A **video controller card**, also called a **video card** or **display adapter**, might have RAM on it that the CPU must communicate with as the CPU passes data to the video card to be sent to the monitor. In addition to RAM, the video card also contains some programming to manage video stored in ROM chips on the card. This video BIOS requires some memory addresses so that the CPU can access this BIOS.

Lastly, expansion cards most often contain ROM chips with BIOS to control the peripheral devices they support. In addition, some cards might have RAM chips that also need memory addresses.

How Are Memory Addresses Assigned?

Memory addresses are, for the most part, assigned during the boot process. Recall from Chapter 1 that the boot process begins with startup BIOS on the ROM BIOS chip. This program in ROM is assigned addresses, as is the portion of ROM that contains the system BIOS. Video ROM and RAM get their address assignments early in the boot process so that the startup BIOS has the use of video while booting. Any other ROM or RAM chips on expansion cards can also request memory addresses during the boot process. Also during booting, addresses are assigned to RAM stored on DIMMs and SIMMs on the system board. Some of this RAM is used to hold the OS, device drivers, and data used by both. Most of this RAM is not used until applications and their data are loaded after booting.

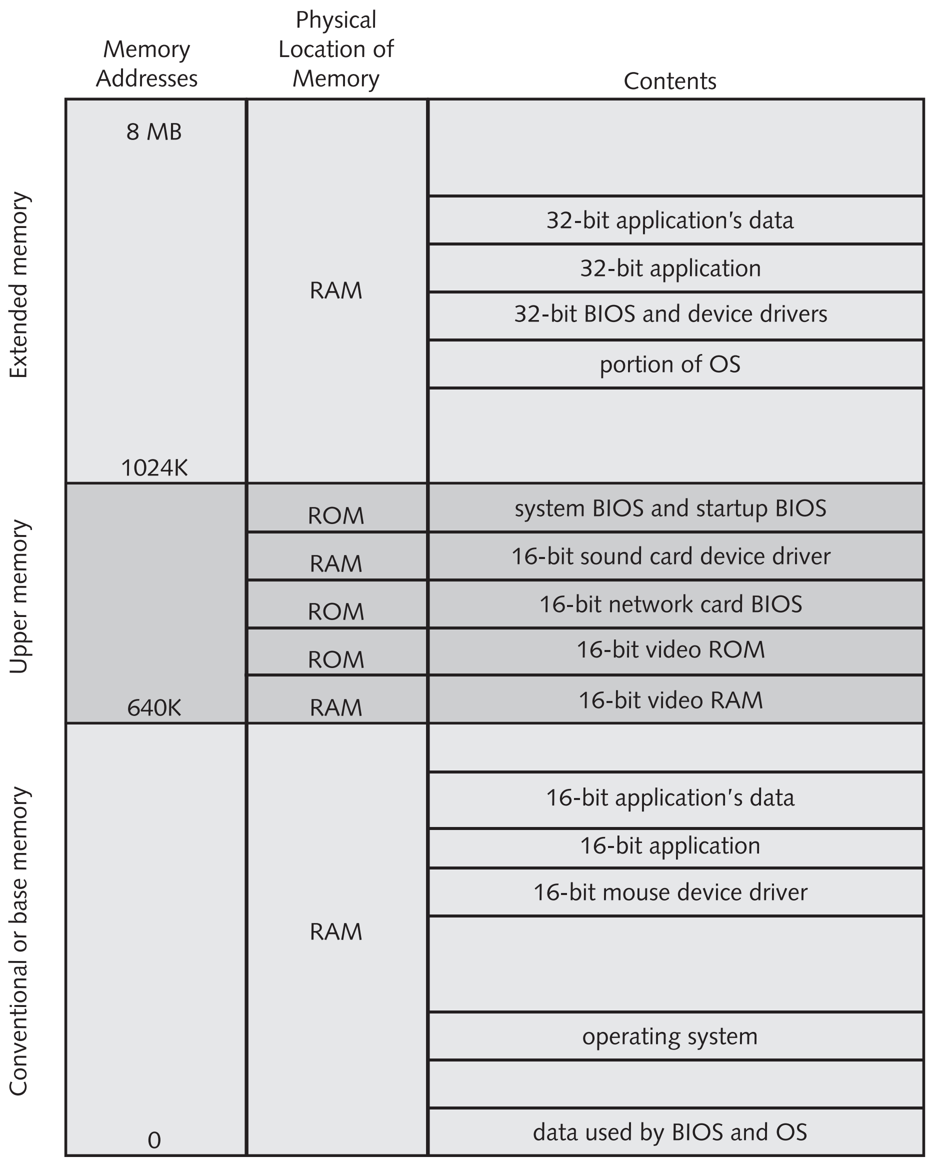


Figure 2-14 Memory map showing how ROM and RAM, on and off the system board, might be mapped to memory addresses

A+ CORE 2.2 OS 1.1

BIOS and Device Drivers That Request Specific Memory Addresses

Today's BIOS and software don't expect a specific group of addresses to be assigned to them. However, older BIOS and device drivers designed to run in a DOS real-mode environment sometimes required a specific group of memory addresses in order to load. You'll learn more about this in Chapter 4, but for now just know that BIOS or real-mode device drivers may

A+ CORE 2.2 OS 1.1

only work if they are given a specific group of addresses. These addresses are usually in the upper memory address range between 640K and 1024K (see Table 1-8 in Chapter 1 for the memory divisions under DOS).

Shadowing ROM

Before leaving our discussion of memory, let's review one other interesting use of ROM and RAM, discussed in Chapter 1, that helps clarify how memory addresses are used. Remember that sometimes system BIOS programs are copied into RAM because reading from RAM chips is generally faster than reading from ROM chips. The process of copying programs from ROM to RAM for execution is called **shadowing ROM** or just **shadow RAM**. If the ROM programs are executed directly from the ROM chips, the memory addresses are assigned to this ROM. If the programs are first copied to RAM and then executed, the same memory addresses are assigned to this area of RAM. Either way, the instructions work the same way.

Input/Output Addresses

Another system resource that is made available to hardware devices is input/output addresses or I/O addresses. **I/O addresses**, or **port addresses**, sometimes simply called **ports**, are numbers that the CPU can use to access hardware devices, in much the same way it uses memory addresses to access physical memory. As you saw earlier in the chapter, the address bus on the system board sometimes carries memory addresses and sometimes carries I/O addresses. If the address bus has been set to carry I/O addresses, then each device is "listening" to this bus. See Figure 2-15. If the address belongs to it, then it responds; otherwise it ignores the request for information. In short, the CPU "knows" a hardware device as a group of I/O addresses. If it wants to know the status of a printer or a floppy drive, for example, it passes a particular I/O address down the address bus on the system board.

A+ CORE 2.2

A few common assignments for I/O addresses are listed in Table 2-6. Because IBM made many address assignments when the first PC was manufactured in the late 1970s, common devices such as a hard drive, a floppy drive, or a keyboard have no problem with I/O addresses. Their BIOS can simply be programmed to use these standard addresses. Devices such as scanners or network cards that were not assigned I/O addresses in the original IBM list can be configured to use more than one group of addresses, depending on how they are set up during either the installation process or the boot process. More about this in Chapter 9.

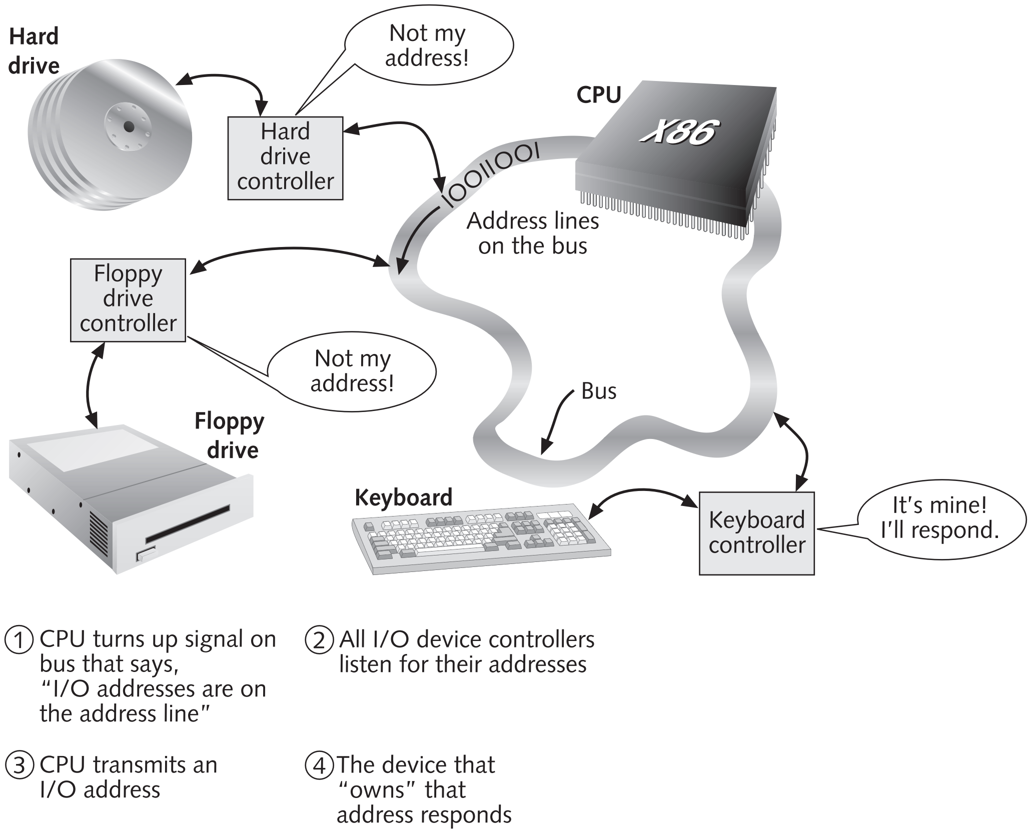

Hard drive
Not my address!
Hard drive controller
CPU
X86
IOOIIOOI
Address lines on the bus
Floppy drive controller
Not my address!
Floppy drive
Bus
Keyboard
Keyboard controller
It's mine! I'll respond.
1 CPU turns up signal on bus that says, "I/O addresses are on the address line"
2 All I/O device controllers listen for their addresses
3 CPU transmits an I/O address
4 The device that "owns" that address responds

Figure 2-15 I/O address lines on a bus work much like an old telephone party line; all devices "hear" the addresses, but only one responds

A+CORE 2.2

Table 2-6 Interrupt Request Numbers and I/O addresses for devices

IRQ	I/O Address	Device
0	040–05F	System timer
1	060–06F	Keyboard controller
2	0A0–0AF	Access to IRQs above 7
3	2F8–2FF	COM2 (covered in Chapter 9)
3	2E8–2EF	COM4 (covered in Chapter 9)
4	3F8–3FF	COM1 (covered in Chapter 9)
4	3E8–3EF	COM3 (covered in Chapter 9)
5	278–27F	Sound card or parallel port LPT2 (covered in Chapter 9)
6	3F0–3F7	Floppy drive controller
7	378–37F	Printer parallel port LPT1 (covered in Chapter 9)
8	070–07F	Real time clock

A+CORE 2.2

Table 2-6 Interrupt Request Numbers and I/O addresses for devices (continued)

IRQ	I/O Address	Device
9–10		Available
11		SCSI or available (covered in Chapter 6)
12	238–23F	System-board mouse
13	0F8–0FF	Math coprocessor
14	1F0–1F7	IDE hard drive (covered in Chapter 6)
15	170–170	Secondary IDE hard drive or available (covered in Chapter 6)

Direct Memory Access (DMA) Channels

Another system resource used by hardware and software is a **DMA** (**direct memory access**) channel, a shortcut method whereby an I/O device can send data directly to memory, bypassing the CPU. A chip on the system board contains the DMA logic and manages the process. In earlier computers there were four channels numbered 0, 1, 2, and 3. Later, channels 5, 6, and 7 were added when the 16-bit ISA bus was introduced. You can see the lines on the bus needed to manage these channels in Figures 2-9 and 2-10. Each channel requires two lines to manage it, one for the DMA controller to request clearance from the CPU and the other used by the CPU to acknowledge that the DMA controller is free to send data over the data lines without interference from the CPU.

DMA channel 4 is used as IRQ 2 was used, to connect to the higher IRQs. In Figure 2-16, note that DMA channel 4 cascades into the lower DMA channels. DMA channels 0–3 use the 8-bit ISA bus, and DMA channels 5, 6, and 7 use the 16-bit ISA bus. This means that the lower four channels provide slower data transfer than the higher channels, because they don't have as many data paths available. Also, an 8-bit expansion card that is only using the 8-bit ISA bus cannot access DMA channels 5, 6, or 7 because it can't get to these pins on the extended expansion slot.

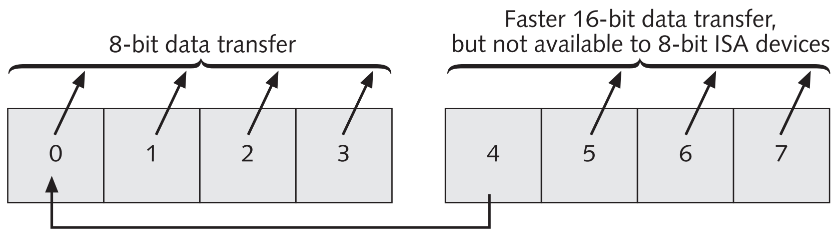

8-bit data transfer
Faster 16-bit data transfer, but not available to 8-bit ISA devices
0
1
2
3
4
5
6
7

Figure 2-16 DMA channel 4 is not available for I/O use because it is used to cascade into the lower four DMA channels

Some devices, such as a hard drive, are designed to use DMA channels, and others, such as the mouse, are not. Those that use the channels might be able to use only a certain channel, say channel 3, and no other. Or the BIOS might have the option of changing a DMA channel number to avoid conflicts with other devices. Conflicts occur when more than one device uses

A+CORE 2.2

the same channel. DMA channels are not as popular as they once were because their design makes them slower than newer methods. However, slower devices such as floppy drives, sound cards, and tape drives may still use DMA channels.

Tying It All Together

Let's take one more look at how system resources are allocated and used to manage hardware devices, and then we'll examine a comprehensive example to see how all the parts come together to perform a task. At startup, a hardware device is assigned (1) an IRQ by which it can signal the CPU that it needs attention, (2) some I/O addresses by which the CPU and the device can communicate, (3) some memory addresses that indicate where the program to manage the device can be stored, and (4) perhaps a DMA channel to speed up sending its data to memory. Later, when a hardware device needs attention from the CPU, the device raises its IRQ line to the CPU. When the CPU senses the IRQ, it stops what it is doing and handles the interrupt. The CPU looks at the location in memory where the device driver or BIOS program that services the device is stored. The CPU then executes this program, which will use the I/O addresses to communicate with the device.

Hardware Interrupts

And now for our comprehensive example of a hardware interrupt. For the keyboard shown in Figure 2-17 the process works like this:

Step 1. A key is pressed on the keyboard. The keyboard controller raises its assigned IRQ to the CPU, saying, "I need attention." The CPU sees the IRQ, acknowledges it, and turns its attention to servicing it. By sending the acknowledgment, it is requesting the device controller sends a number called an interrupt (abbreviated INT) that tells the CPU what service the device needs. There is more about interrupts coming up.

Step 2. The keyboard controller sends INT 9 to the CPU. The CPU uses this value to locate the program to handle the interrupt. This program, which may be either BIOS or a device driver, is called an **interrupt handler**.

Step 3. The CPU looks to a table in RAM called the **interrupt vector table**, or **vector table**, that contains a list of memory address locations of interrupt handlers. The INT value passed to the CPU by the controller points to the correct row in the table where the memory addresses in which the instructions to service the keyboard (a portion of system BIOS) are stored.

Step 4. The CPU looks to the location in memory of the request handler and begins to follow the instructions there.

Step 5. The CPU, following the interrupt handler instructions, processes the keystroke.

TIP

The BIOS and operating system initialize the interrupt vector table during booting, but later another program can modify the vector table to change the interrupt handler location to execute another program instead. This is a common method that a virus uses to propagate itself.

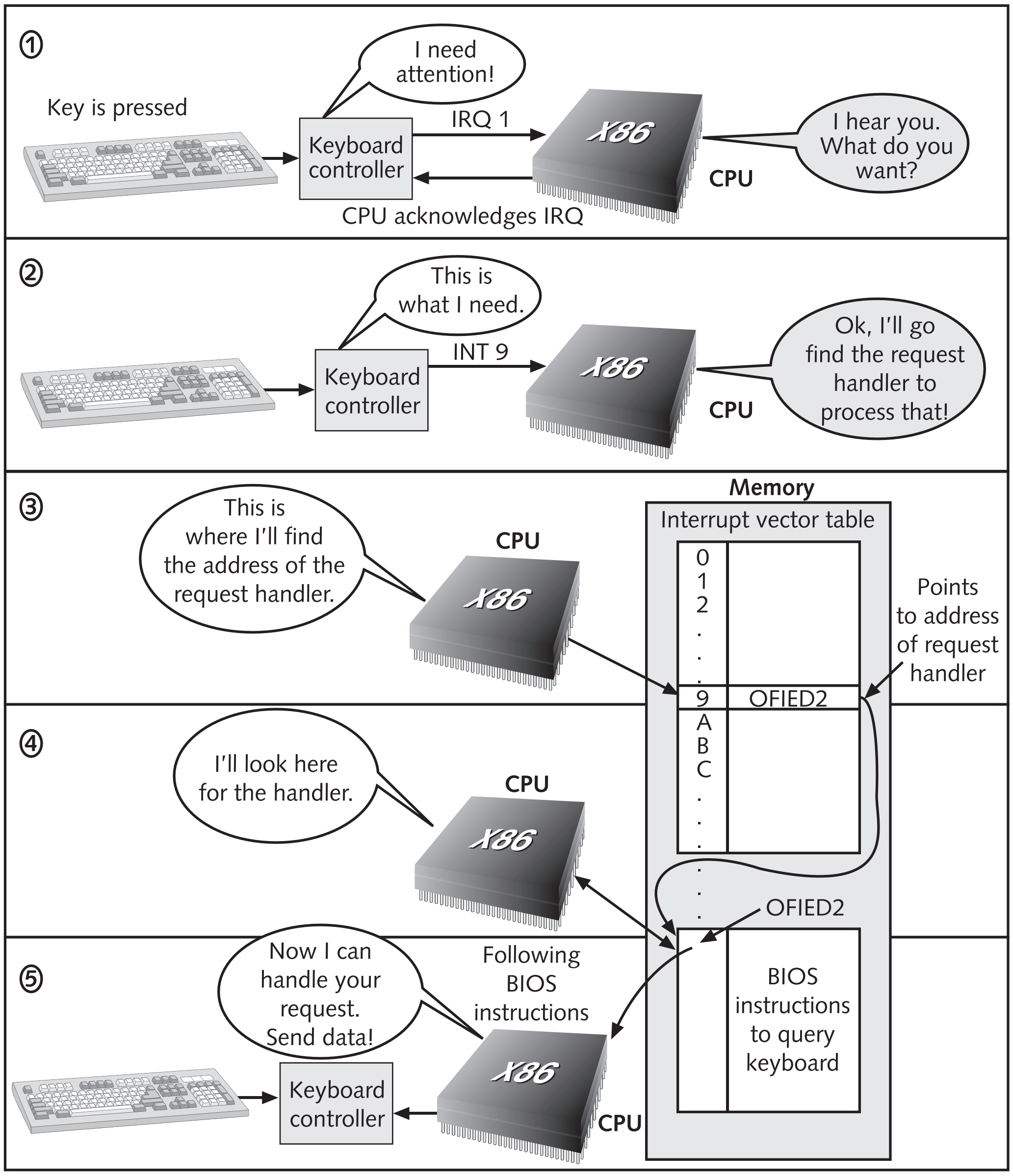
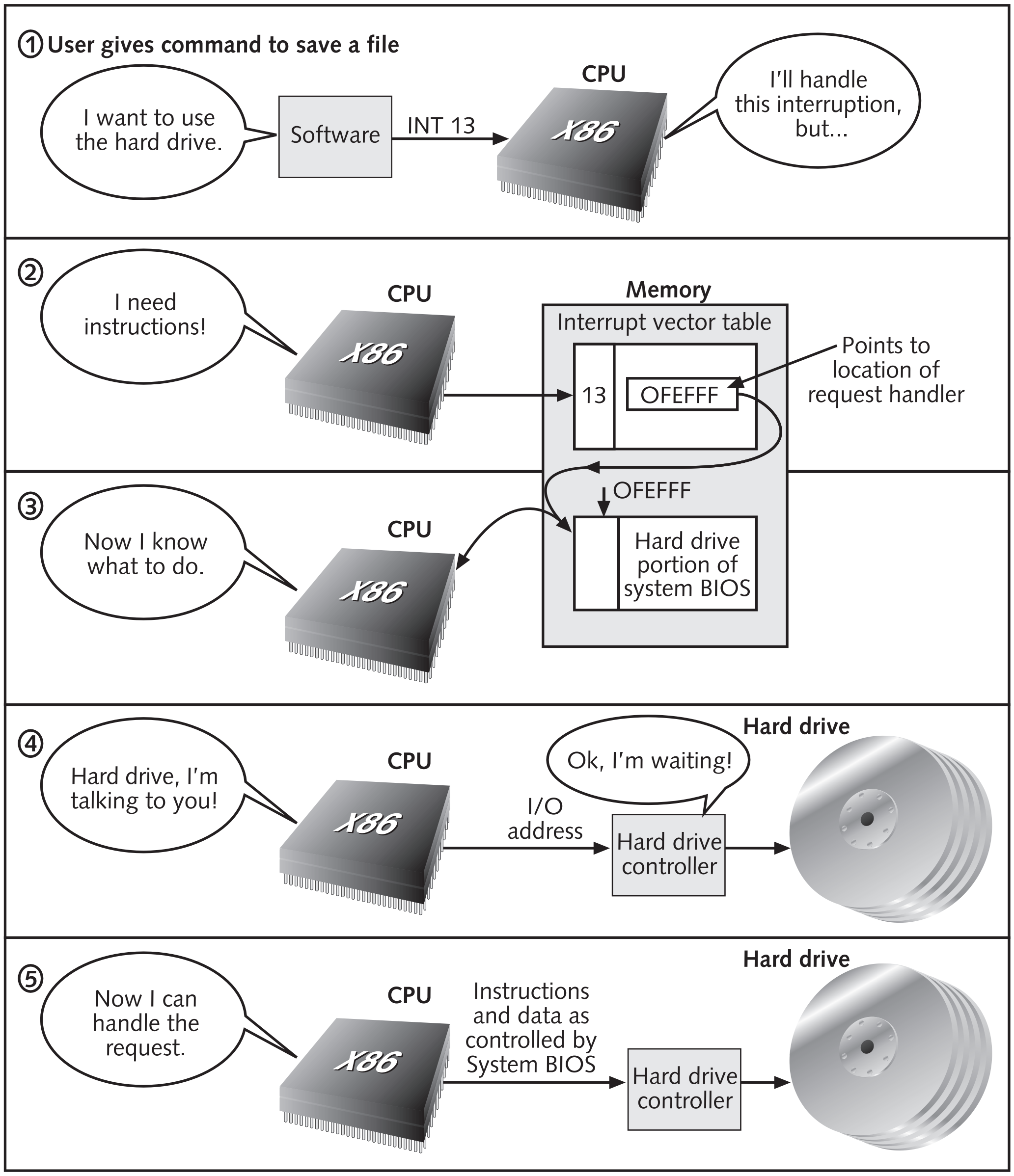

①
Key is pressed
I need attention!
IRQ 1
Keyboard controller
CPU acknowledges IRQ
X86
CPU
I hear you. What do you want?
②
This is what I need.
INT 9
Keyboard controller
X86
CPU
Ok, I'll go find the request handler to process that!
③
This is where I'll find the address of the request handler.
CPU
X86
Memory
Interrupt vector table
0 1 2 . . . 9 A B C . . . .
OFIED2
Points to address of request handler
④
I'll look here for the handler.
CPU
X86
OFIED2
⑤
Now I can handle your request. Send data!
Following BIOS instructions
BIOS instructions to query keyboard
X86
Keyboard controller
CPU

Figure 2-17 The story of a hardware interrupt, where the request handler is BIOS

Software Interrupts

In this last example, two of the four system resources were used (memory addresses and an IRQ). The keyboard controller used an IRQ to initiate communication, which means that this is an example of a hardware interrupt. When software initiates communication, such as when the user of word-processing software gives the command to save a file to the hard drive, this is known as a **software interrupt**, which is demonstrated in Figure 2-18.

Both hardware and software interrupts use the same numeric INT values to communicate their requests to the CPU. Appendix G contains a list of INT values.



① User gives command to save a file
I want to use the hard drive.
Software
INT 13
CPU
X86
I'll handle this interruption, but...
②
I need instructions!
CPU
X86
Memory
Interrupt vector table
13
OFEFFF
Points to location of request handler
③
Now I know what to do.
CPU
X86
OFEFFF
Hard drive portion of system BIOS
④
Hard drive, I'm talking to you!
CPU
X86
I/O address
Ok, I'm waiting!
Hard drive controller
Hard drive
⑤
Now I can handle the request.
CPU
X86
Instructions and data as controlled by System BIOS
Hard drive controller
Hard drive

Figure 2-18 The story of a software interrupt

The interrupt value for a call to the hard drive for I/O interaction is INT 13. (1) The CPU receives the software interrupt, INT 13, and (2) turns to the interrupt vector table, using the INT value to locate the correct request handler. Next, (3) the CPU locates the handler in memory, (4) alerts the hard drive that instructions are forthcoming by sending its I/O address over the address bus, and (5) follows the instructions of the request handler, in this case, system BIOS, to manage the hard drive.

Incidentally, in Chapter 6 you will see how this process of software using INT 13 to interface with a hard drive has led to problems: more sophisticated methods of hard drive interface have been developed, but some legacy software still expects to use the INT method. Complex workarounds have become the solution to making hard drive interfaces backward-compatible.

Configuration Data and How It Is Stored

Recall from Chapter 1 that **configuration data**, or setup data, is stored in the computer in one of these devices: DIP switches, CMOS setup chip, or jumpers. This setup information can contain additional information such as system resources that a device requires or system date and time. This section takes a closer look at these storage methods.

Storing configuration information by physically setting DIP switches or jumpers on the system board or peripheral devices is inconvenient because it often requires opening the computer case to make a change. A more convenient method is to use the CMOS chip to hold the information in the small amount of RAM on the chip. A program in BIOS can then be used to easily make changes to setup.

Setup Data Stored by DIP Switches

Many older computers and a few newer ones store setup data using DIP switches on the system board, as shown in Figure 2-19. A DIP (dual in-line package) switch is a switch that has an ON position and an OFF position. ON represents binary 1, and OFF represents binary 0. If you add or remove equipment, you can communicate that to the computer by changing a DIP switch setting. When you change a DIP switch setting, use a pointed instrument, such as a ballpoint pen, to push the switch. Don't use a graphite pencil, because graphite conducts electricity. Pieces of graphite dropped into the switch can damage it.

Setup Data Stored on a CMOS Chip

Most configuration data in newer computers is stored in a CMOS microchip that is battery powered when the system power is off (see Figure 2-20). The advantage of a CMOS chip over other types of chips is that CMOSs require very little electricity to hold data. A small trickle of electricity from a nearby battery enables the CMOS chip to hold the data even while the main power to the computer is off.

On older computers (mostly IBM 286 PCs built in the 1980s), changes are made to the CMOS setup data using a setup program stored on a floppy disk. One major disadvantage of this method (besides the chance that you might lose or misplace the disk) is that the disk drive must be working before you can change the setup. An advantage of this method is that you can't change the setup unintentionally.

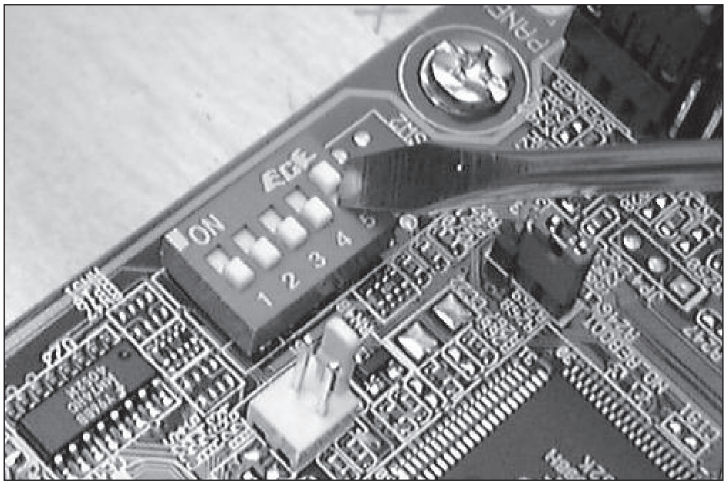


Figure 2-19 DIP switches are sometimes used to store setup data on system boards

On newer computers, you usually change the data stored in the CMOS chip by accessing the setup program stored in ROM BIOS.You access the program by pressing a combination of keys during the booting process. The exact way to enter setup varies from one system-board manufacturer to another. See the system-board documentation for the method to enter setup. A message such as the following usually appears on the screen:

```
    Press DEL to change Setup
```

or

```
    Press F8 for Setup.
```

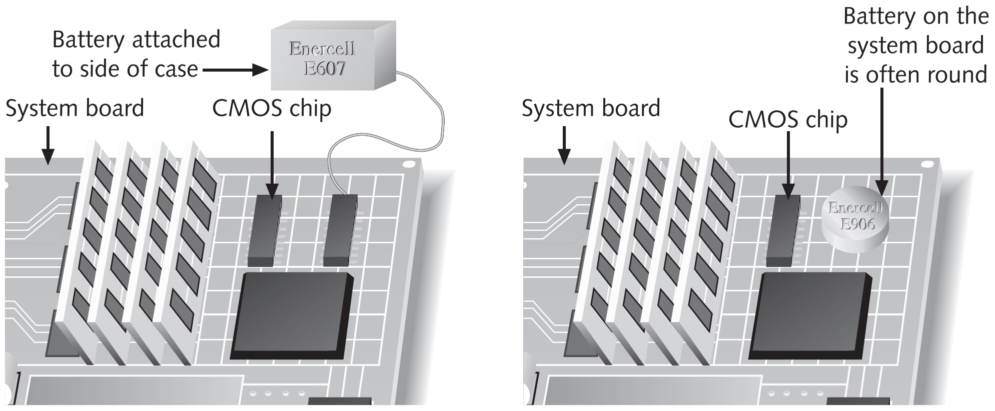

Battery attached to side of case
Enercell E607
System board
CMOS chip
Battery on the system board is often round
System board
CMOS chip
Enercell E906

Figure 2-20 The battery that powers the CMOS chip may be on the system board or attached nearby

When you do so, a setup screen appears with menus and Help features that is often very user- friendly. When you exit the program, you can exit without saving your changes or exit and save your changes to the CMOS chip.

Setup Data Stored by Jumpers

Most computers hold additional configuration information by using jumpers on the system board (see Figure 2-21). A jumper consists of two pins sticking up side by side, with a cover over the two pins making a connection. The two pins and the connection together serve as electrical connectors on the system board. If the pins are not connected with a cover, the setting is considered OFF. If the cover is present, the setting is ON. For older system boards, the presence of cache memory is a typical setting that was communicated to the computer by jumpers. Jumpers also are used to communicate the type and speed of the CPU to the system or to disable a feature on the system board, such as a keyboard wake up. (With this feature enabled, you can press a key to power up the system.) You change the jumper setting by removing the computer case, finding the correct jumper, and then either placing a metal cover over the jumper or removing the cover already there.

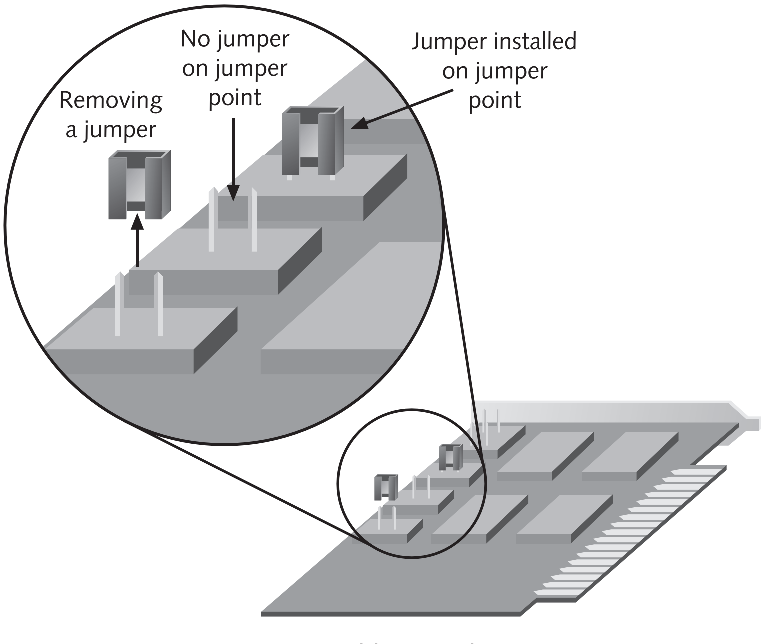


Figure 2-21 Jumpers on an add-on card

A+ CORE 4.4

Passwords Stored on CMOS

Access to a computer can be controlled using **startup passwords**, sometimes called **power-on passwords**. During booting, or startup, the computer asks for a password. If you do not enter the password correctly, the booting process is terminated. The password is stored on the CMOS chip and is changed by accessing the setup screen. Many computers also provide a jumper near the CMOS chip that, when set to "on," causes the computer to "forget" any changes that have been made to default settings stored in CMOS. By jumping these pins, you can disable a password.

PROTECTING DATA, SOFTWARE, AND HARDWARE

Although someone responsible for a computer must understand its operations to solve problems that can occur during use, another objective in maintaining a well-functioning system is to protect the data, software programs, and hardware from harm. Despite your best efforts, data is sometimes lost, software stops functioning correctly, and hardware fails. By taking a few practical measures, however, you can avoid some common causes of loss. Here are some practical guidelines that all computer users should follow. However, it is usually the computer support person who is called upon to save the day if something is lost. As a computer support person, you should take the time to train yourself and your users about preventive maintenance and practical precautions.

Saving and Restoring Setup Information in CMOS

A+CORE 4.4

Because the information stored in CMOS is so important to the successful operation of a computer, keep a backup of that information in case the CMOS data is lost. You can use several utility software programs to back up setup information to a disk you can use to recover lost setup information. In this chapter, we introduce two programs: Nuts & Bolts, which comes on the CD accompanying this book, and Norton Utilities, another popular PC utility program. Setup information on a PC can be lost if the battery dies or is replaced. Information also can be lost when errors occur on the system board, and a user can accidentally change setup without realizing how significant the consequences can be. Possible errors and events that might indicate that setup information is lost are:

- The battery is discharged. An early indication of a weak battery is that the PC loses the correct date and time when turned off.
- Message at startup says, "Hardware information is lost" or "CMOS checksum error." These errors can be caused by a dead battery or a poorly connected battery.
- The battery has been replaced. After the battery is replaced, restore the CMOS settings using a rescue disk.

Saving Setup Information Using Nuts & Bolts

Follow the steps below to create a rescue disk using Nuts & Bolts.

To save setup information to disk:

1. Select **Start**, **Programs**, **Nuts & Bolts**. The list of utilities included with the software appears, as shown in Figure 2-22. Select **Rescue Disk**.
2. The Welcome to Rescue Disk screen is displayed. Click **Next** to continue.
3. From the next screen (see Figure 2-23), you can choose to format the disk that will become your rescue disk or plan to use a previously formatted disk.
4. Click **Next**. Nuts & Bolts prompts you to insert the disk in drive A and creates the rescue disk.

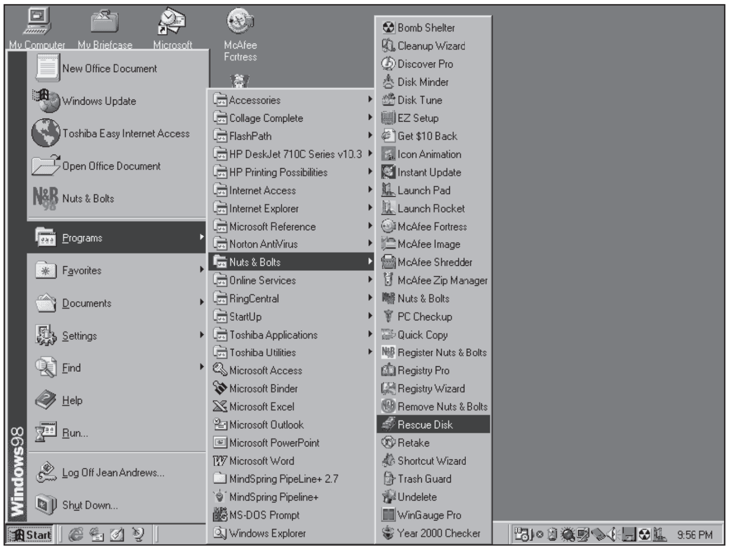
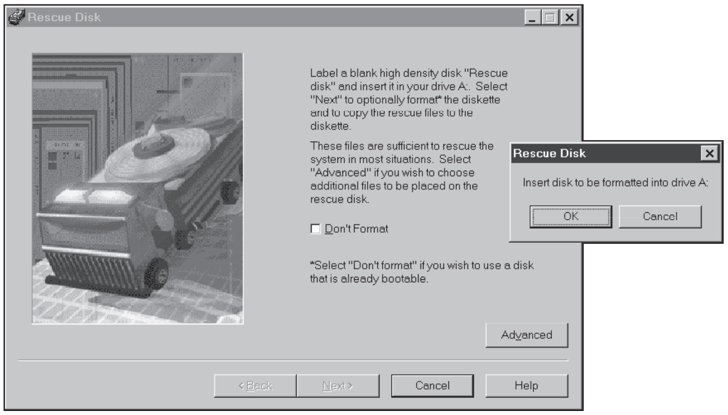


Figure 2-22 The Nuts & Bolts suite of utilities that is on the CD that accompanies this book

5. Click **Finish** when the process is complete.

   The disk contains many files that can help rescue a system with problems. The file CMOS.DAT contains a backup of the data in CMOS setup.


Rescue Disk

Label a blank high density disk "Rescue disk" and insert it in your drive A:. Select "Next" to optionally format* the diskette and to copy the rescue files to the diskette.

These files are sufficient to rescue the system in most situations. Select "Advanced" if you wish to choose additional files to be placed on the rescue disk.

☐ Don't Format

*Select "Don't format" if you wish to use a disk that is already bootable.

Rescue Disk

Insert disk to be formatted into drive A:

Figure 2-23 The Nuts & Bolts Rescue Disk utility which, among other things, will save CMOS data to disk

To use a rescue disk to restore setup information when it is lost:

1. Insert the disk labeled Rescue Disk 1, which is a bootable disk, into the disk drive, turn the PC off and, after a pause, back on. This disk boots to drive A and provides an A prompt.
2. At the A prompt, type **RESCUE**, and then press **Enter**.

3. Select **CMOS Information** from the items to restore, and then press **Alt+R** to begin the process. Follow the instructions on screen.
4. Remove the rescue disk from the drive and reboot. CMOS information should now be restored. Return the rescue disks to a safe place.

Saving Setup Information Using Norton Utilities

Follow the steps below to create a rescue disk using Norton Utilities for Windows 9x. (Other uses of this software are found in later chapters.)

To save setup information to disk:

1. Select **Start**, **Programs**, **Norton Utilities**. The list of utilities included with the software appears, as shown in Figure 2-24. Select **Rescue Disk**.

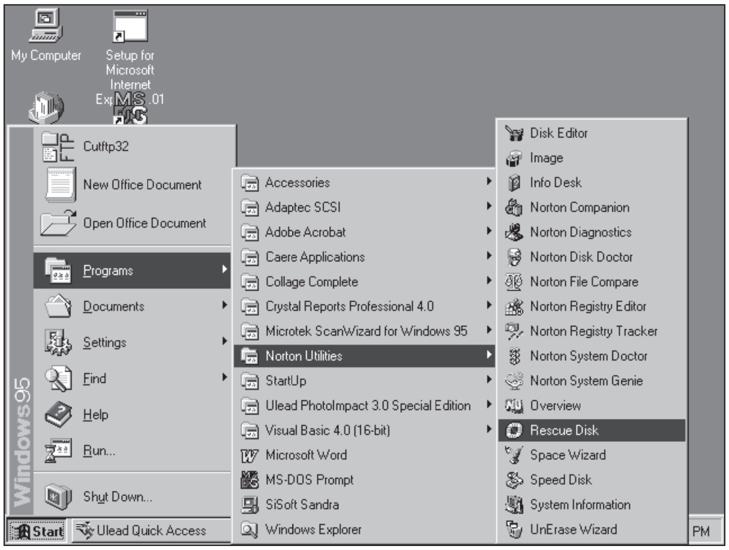
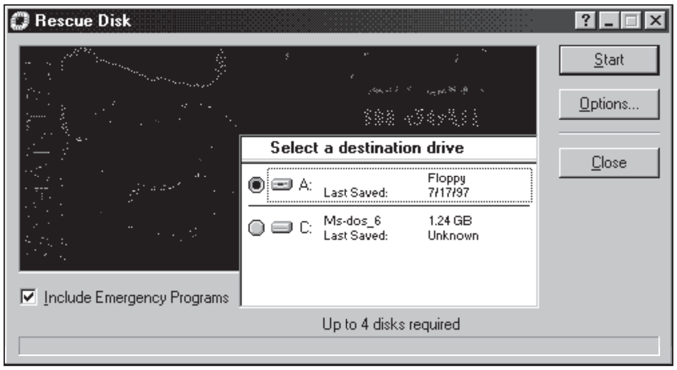


Figure 2-24 Norton Utilities lists a suite of utility programs

2. The Norton Utilities Rescue Disk window opens, as shown in Figure 2-25. To see the list of items that Norton Utilities saves to the set of rescue disks, click **Options**, which opens the dialog box shown in Figure 2-26. We have already introduced some of the items on the list, and will discuss others in future chapters. For now, notice the item CMOS Information, which will be saved to disk. Make sure that **Rescue** is checked to be included in the list, then click **OK** to return to the opening window.
3. Check that the appropriate disk drive is selected as the destination drive, and then click **Start** to begin the process.
4. Norton Utilities prompts you to insert the first disk.
5. When the information is saved to disk, the process ends and you are prompted to properly label and store the disk(s).



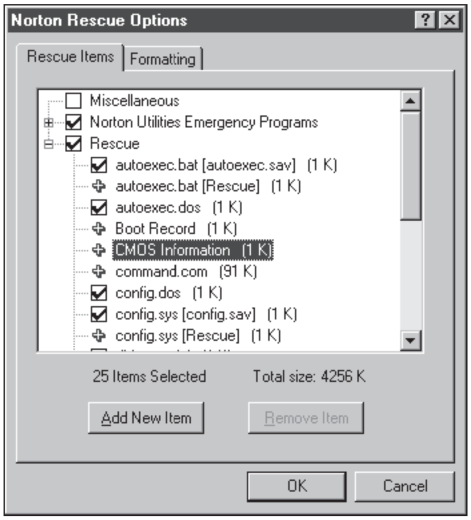
Select a destination drive

● A: Last Saved:	Floppy	7/17/97
○ C: Ms-dos_6 Last Saved:	1.24 GB	Unknown

☑ Include Emergency Programs

Up to 4 disks required

Figure 2-25 Creating a set of rescue disks using Norton Utilities



- ☐ Miscellaneous
- ☑ Norton Utilities Emergency Programs
- ☑ Rescue
  - ☑ autoexec.bat [autoexec.sav] (1 K)
  - ✚ autoexec.bat [Rescue] (1 K)
  - ☑ autoexec.dos (1 K)
  - ✚ Boot Record (1 K)
  - ✚ CMOS Information (1 K)
  - ✚ command.com (91 K)
  - ☑ config.dos (1 K)
  - ☑ config.sys [config.sav] (1 K)
  - ✚ config.sys [Rescue] (1 K)

25 Items Selected Total size: 4256 K

Figure 2-26 List of items to be stored on the rescue disks

To use a rescue disk to restore setup information when it is lost:

1. Insert the disk labeled Rescue Disk 1, which is a bootable disk, into the disk drive, turn the PC off and, after a pause, back on. This disk boots to drive A and provides an A prompt.
2. At the A prompt, type **RESCUE**, and then press **Enter**.
3. Select **CMOS Information** from the items to restore, and then press **Alt+R** to begin the process. Follow the instructions on screen.
4. Remove the rescue disk from the drive and reboot. CMOS information should now be restored. Return the rescue disks to a safe place.

Keeping OS Rescue Disks

A+ OS 2.3

An OS rescue disk, which allows you to boot a computer, is essential for every computer. You will learn later in the book about other software that should go on a bootable disk, but, at the least, a bootable disk should contain enough software to load the OS. A rescue disk can be created for DOS using the FORMAT A:/S command. Then copy the AUTOEXEC.BAT and CONFIG.SYS files from the root directory of your hard drive to the disk.

For Windows 9x, use Explorer to create a bootable disk. If you install Windows 95 from a CD rather than from floppy disks, be sure to include a DEVICE= line in a CONFIG.SYS file to load the driver for the CD-ROM and include the CD-ROM driver on the disk. A typical line might look like this (details of this command line are discussed in Chapter 10):

```
DEVICE=C:\CDSYS\SLCD /D:MSCD001
```

In either case, test the disk to make sure it works. To do this, insert the disk in the drive, hard boot, and verify that your OS does load. For Windows 95, verify that you can access your CD-ROM.

Windows 9x also offers a much better choice for creating a rescue disk. Follow these directions to create an emergency rescue disk for Windows 9x.

1. Click **Start** on the Taskbar, point to **Settings**, and then click **Control Panel**.
2. In the Control Panel window, double-click the **Add/Remove Programs** icon.
3. Click the **Startup Disk** tab, and then click the **Create Disk** button. The disk will then be created.

More details about rescue disks are included in later chapters.

Backing Up Your Hard Drive

How valuable is your data? How valuable is your software? In many cases, the most valuable component on the desktop is not the hardware or the software, but the data. Think about each computer you support. What would happen if the hard drive failed? Now create backups to prepare for just that situation. A **backup** is an extra copy of a file or files made to a different location or storage media.

Your backup policy depends on what you are backing up. If you use your PC to interface with a server, for example, and all data is stored on the server and not on the PC, then obviously, you should only back up software. (The person responsible for the server should back up the data.) If you keep original software disks and CDs in a safe place, and if you have multiple copies of them, you might decide not to back up the software. In this case, if a hard drive fails, your chore is to reload several software packages.

However, if you maintain a large database on the hard drive of your PC, you need to seriously consider a sophisticated backup method. Suppose this database is quite large and is edited daily, several times a day. If this database is lost, so are thousands of labor hours. Plan for the worst case! A good tape backup system is probably in order. Maintain 5 or 10 tapes, on which a complete backup of the database is made each night.

Even if the hard drive contains only a few important word-processing files, make backups. Never keep an important file on only one medium. Make a duplicate copy to a disk, to a file server, or to tape backup. Consider keeping some of your backups in an off-site location.

When protecting your data, the best plan is to back up. Assume that one day the primary medium you are using for this data will fail. Know the steps you will take when that happens and prepare for it.

Documentation

Make sure to keep the documentation that goes with your hardware and software. Suppose someone decides to tinker with a PC for which you are responsible and changes a jumper on the system board, but no longer remembers which jumper he or she changed. The computer no longer works, and the documentation for the board is now invaluable. If documentation is lost or misplaced, a simple job of reading the settings for each jumper and checking them on the board can become a long and tedious research task. Keep the documentation well-labeled in a safe place. If you have several computers to maintain, you might consider a filing system for each computer. Another method is to tape a cardboard folder to the inside top of the computer case and safely tuck the hardware documentation here. This works well if you are responsible for several computers spread over a wide area.

Damage from Electricity

Computers and data can be destroyed by two kinds of electricity—static electricity, known as **ESD** (**electrostatic discharge**), and power spikes, including lightning. You can do many practical things to protect from both.

ESD is most dangerous when the computer case is off. Never touch the inside of a computer without first taking precautions to protect the hardware from static electricity. Make sure both you and the computer are grounded before you touch anything inside. *Never* touch the inside of the computer when the computer is turned on. The Introduction to this book presents more extensive instructions on how to protect your computer when you work on it. Be sure to read this material before starting to work on your computer.

You can use several devices to protect the computer against electrical surges and lightning. These devices are discussed in Chapter 11.

Chapter Summary

- Four system resources that aid in the communication between hardware and software are I/O addresses, IRQs, DMA channels, and memory addresses.
- An IRQ is a line on a bus that a device uses to alert the CPU that it needs servicing.
- A DMA channel provides a shortcut for a device to send data directly to memory, bypassing the CPU.
- Memory addresses are hex numbers, often written in segment/offset form, assigned to RAM and ROM so that the CPU can access both.
- The CPU sends a device's I/O address over the address bus when it wants to initiate communication with the device.

- Startup BIOS performs a power-on self test (POST) that surveys and tests hardware, examines setup information, and assigns system resources to the hardware. Startup BIOS then begins the process of loading the OS.
- When the OS loads from a hard drive, the first program BIOS executes is the master boot record (MBR), which executes the OS boot record, which, for DOS, attempts to find IO.SYS and MSDOS.SYS on the hard drive.
- IO.SYS and MSDOS.SYS together with COMMAND.COM form the kernel, or core, of DOS.
- AUTOEXEC.BAT and CONFIG.SYS are two files that contain commands used to customize the OS load process.
- After DOS is loaded, Windows 3.x can be executed from the WIN command stored in AUTOEXEC.BAT.
- Windows 9x uses Plug and Play to help install devices and assign resources to them during the boot process.
- In Windows 9x, the file MSDOS.SYS is a text file that contains parameters to customize the boot.
- One bus used on early PCs is the 8-bit ISA bus, which was later improved to become the 16-bit ISA bus that is still used on PCs today.
- Under Windows 9x, use Device Manager to find out how system resources have been allocated in your system.
- A hardware interrupt is initiated by a hardware device sending an IRQ to the CPU. A software interrupt is initiated by the software sending an interrupt number (INT) to the CPU.
- Setup data can be stored on the CMOS chip on the system board and by DIP switches or jumpers.
- There are several utility programs, such as Nuts & Bolts and Norton Utilities, that can save a copy of CMOS setup to disk. Use these to back up setup information.
- For safety's sake, back up important information on your hard drive, keep documentation in a safe place, and protect your computer against both static electricity and electrical power surges.

Key Terms

AUTOEXEC.BAT — One startup file on an MS-DOS computer. It tells the computer what commands or programs to execute automatically after bootup.

Back up, Backup — When used as a verb, to make a duplicate copy of important files or data. When used as a noun, refers to the file created when backing up. Backups can be made by saving a file with a different name or by copying files to a different disk or to a tape drive.

Cold Boot — *See* Hard boot.

Configuration data — Also called setup information. Information about the computer's hardware, such as what type of hard drive or floppy drive is present, along with other detailed settings.

Display adapter — *See* Video controller card.

DMA (direct memory access) controller chip — A chip that resides on the system board and provides channels that a device may use to send data directly to memory, bypassing the CPU.

ESD (electrostatic discharge) — Another name for static electricity, which can damage chips.

Hard boot — Restart the computer by turning off the power or by pressing the Reset button. Also called cold boot.

Hardware interrupt — An event caused by a hardware device signaling the CPU that it requires service.

Hidden file — A file that is not displayed in a directory list. To hide or display a file is one of the file's attributes kept by the OS.

Interrupt handler — A program (either BIOS or a device driver), that is used by the CPU to process a hardware interrupt.

Interrupt vector table — A table that stores the memory addresses assigned to interrupt handlers. Also called a vector table.

I/O addresses — Numbers that are used by devices and the CPU to manage communication between them.

IRQ (interrupt request number) — A line on a bus that is assigned to a device and is used to signal the CPU for servicing. These lines are assigned a reference number (for example, the normal IRQ for a printer is IRQ 7).

On-board BIOS – *See* System BIOS.

Polling — A process by which the CPU checks the status of connected devices to determine if they are ready to send or receive data.

POST (power-on self test) — A self-diagnostic program used to perform a simple test of the CPU, RAM, and various I/O devices. The POST is performed when the computer is first turned on and is stored in ROM-BIOS.

Power-on password — *See* Startup password.

Shadow RAM, Shadowing ROM — ROM programming code copied into RAM to speed up the system operation, because of the faster access speed of RAM.

Soft boot — To restart a PC by pressing three keys at the same time (Ctrl, Alt, and Del). Also called warm boot.

Software interrupt — An event caused by a program currently being executed by the CPU signaling the CPU that it requires the use of a hardware device.

Startup password — A password that a computer requires during the boot process used to gain access to the PC. Also called power-on password.

System BIOS — Basic input/output system chip(s) residing on the system board that control(s) normal I/O to such areas as system memory and floppy drives. Also called on-board BIOS.

TSR (terminate-and-stay-resident) — A program that is loaded into memory but is not immediately executed, such as a screen saver or a memory-resident antivirus program.

Vector table — *See* Interrupt vector table.

Video controller card — An interface card that controls the monitor. Also called video card or display adapter.

Warm boot — *See* Soft boot.

Review Questions

1. List four system resources that software uses to manage hardware.
2. What Windows 9x utility allows you to see the IRQ assignments made to devices?
3. What must happen to a program that is stored on a hard drive before it can be executed?
4. Name some items that might be stored in physical RAM on a system board.
5. Name one system resource that a video card most likely will not need.
6. Where in memory are device drivers most often stored?
7. Is a mouse more likely to be controlled by a device driver or by system BIOS?
8. Name one device that is likely to be controlled by system BIOS.
9. Why is programming code that is stored in ROM BIOS sometimes copied to RAM? What is this process called?
10. How is a software interrupt initiated?
11. How is a hardware interrupt initiated?
12. Describe a request handler. Where in memory can you find a list of addresses where request handlers are located?
13. When the mouse initiates a hardware interrupt to the CPU, how does the CPU know where to find a program to service the mouse?
14. If memory addresses are used by the CPU to access memory, then what are I/O addresses used for?
15. What is the I/O address range for the keyboard?
16. Why are DMA channels not as popular as they once were with high-speed devices?
17. Name a device that uses polling in order to be serviced by the CPU.
18. List in detail the steps that happen when you press a key at the keyboard in order for the keystroke to be communicated to the current application.
19. When two expansion cards request the same system resources, what can be done to solve the problem?
20. Describe how Plug and Play BIOS can help resolve resource conflict problems.
21. How do you change a setting that is controlled by a DIP switch?
22. Describe how you access CMOS setup on your PC.
23. How can you add a startup password to your computer?
24. Why is Flash ROM easier to update than regular ROM BIOS?
25. What is a hidden file? Name a DOS file that is hidden.
26. List three commands that might be found in AUTOEXEC.BAT and the purpose of each.

27. How can you load a TSR that you want to remain in memory the entire time a PC is on?
28. Why is it dangerous to edit the CONFIG.SYS file with a word processor?
29. What is the purpose of IO.SYS in Windows 98?
30. Name a utility software that can be used to make a backup copy of CMOS setup information.

PROJECTS

Hands-on Project

Observing the Boot Process Using DOS

1. Use an operational computer with DOS installed. If your computer has a reset button, press it, and then watch what happens. If your computer does not have a reset button, turn it off, wait a few seconds, and then turn it back on. Write down every beep, light on/off, and message on the screen that you notice. Compare your notes to others' to verify that you are not overlooking something.
2. Unplug the keyboard and repeat the steps in Project 1. Write down what happens that is different.
3. Plug the keyboard back in, unplug the monitor, and repeat Project 1 again. After you reboot, plug the monitor in. Did the computer know the monitor was missing?
4. Put a disk that is not bootable in drive A and press the **Reset** button. If you do not have a Reset button, press **Ctrl+Alt+Del** to soft boot. Write down what you observe.
5. Print the AUTOEXEC.BAT and CONFIG.SYS files stored on the hard drive. Use one of the following methods, not Print Screen.

```
C:\> TYPE filename.ext>PRN or C:\> PRINT filename.ext
```

6. Make a bootable disk using either of the two following DOS commands. If the disk is already formatted, but has no files stored on it, use this command:

```
C:\>SYS A:
```

   To format the disk and also make it bootable, use this command:

```
C:\>FORMAT A:/S
```

   Your disk should now contain a boot record, the two hidden files, and COMMAND.COM. Compare the bytes available on the disk to a disk that is not bootable. Calculate how many bytes must be in the two hidden files.
7. Test your bootable disk by inserting it in drive A and doing a soft boot. What prompt do you see on the screen?
8. At the DOS prompt, enter this prompt command as follows (where the space between P and $ can be used to customize the DOS command prompt):

```
PROMPT $P   $G
```

   What prompt did you get? By examining the prompt, guess what $P in the command line accomplishes and what $G accomplishes. Test your theory by changing the PROMPT command, leaving first $P and then $G out of the command line.

9. Using EDIT, create an AUTOEXEC.BAT file on your bootable disk. Create a PROMPT command to include your first name. Test the command by booting from this disk.
10. Without the appropriate PATH command in your active AUTOEXEC.BAT file, you cannot execute software stored on drive C from the A prompt. Test this theory by trying to execute some applications software that you know is stored on your hard drive. For example, if you have WordPerfect on your hard drive, try to execute the software at the A prompt by using the following command:

    ```
    A:\> WP
    ```

    What error did you get? Why?

Using the Internet for Research

Hands-on Project

Microsoft offers a knowledge base of information on all its products. Learning to find information in the knowledge base is important to your success as a PC support person in a Windows environment. Go to the Microsoft support site at support.microsoft.com, follow these directions, and answer these questions:

1. Search for the specific article ID Q151667. Print the article and fill in the following chart for each file that Windows 9x might put in the root folder of the hard drive.

File name	Purpose of the file

2. What is the article ID of an article that describes the contents of the Windows 9x MSDOS.SYS file? Print the first page of this article.
3. List four articles that contain information about the Windows IO.SYS file.
4. Access the root folder of your hard drive and list the files in that folder that belong to Windows 9x. Mark the files that can safely be deleted without affecting Windows 9x functionality.
5. MS-DOS.SYS and IO.SYS must be in the root folder for Windows 9x to boot. Do you see them there? If they are not visible, what command would allow you to see them?

Observing the Boot Process Using Windows 9x

Hands-on Project

Hard boot your PC. If you are using Windows 95, when you see the message, "Starting Windows 95," press **F8**. If you are using Windows 98, press and hold the **Ctrl** key as the OS loads. Write down what happens when you execute each menu choice by booting several times.

Creating a Startup Disk Using Windows 9x

Hands-on Project

Use a startup disk in Windows 9x whenever you have trouble with the OS. Doing this allows you to boot from drive A. Follow these steps to create the disk:

1. Open **Control Panel**.

2. Double-click **Add/Remove Programs**.
3. Click the **Startup Disk** tab.
4. Click **Create Disk**.

After you have created the startup disk, test it by rebooting the computer with the disk still in the drive. Now, using a second disk, create a bootable disk using Explorer. Compare the contents of the two disks.

Using Shareware from the Internet

Refer to the Projects at the end of Chapter 1 for directions on how to download from the Internet the shareware utility SANDRA (System Analyzer Diagnostic and Report Assistant). Find the answers to the following questions using SANDRA and then print a report containing all the answers.

1. What is the name of the device driver file used to manage the serial ports under Windows?
2. What IRQ is your mouse using? What is the name of the mouse driver file?
3. How much conventional or base memory is free at this time?
4. What is the average access time to your hard drive?
5. If you are using Windows 9x, what is the current value of the BootMulti option found in the Msdos.sys file? What is the purpose of the BootMulti option?

Using Device Manager

Using Device Manager of Windows 9x, complete the following:

1. What is the filename and path of the device driver that is used to manage your printer port LPT1?
2. List the IRQs available on your computer, from 0 to 15.

Using MSD

Access MSD as you did in Chapter 1, with your home or lab computer, and then answer the following questions. You can print the appropriate screen in MSD showing the answers.

1. What is the IRQ assigned to the mouse?
2. What, if any, is the COM port used by the mouse?
3. What IRQ is assigned to the floppy disk drive?
4. What is the I/O or port address used by COM 1?
5. IRQ 2 points to, or "cascades" to, a higher IRQ. Which IRQ does it point to?

Using Norton Utilities to Save Setup Information

Using Norton Utilities, create a rescue disk containing setup information, following the directions given in the chapter. After the rescue disk is created, execute the RESCUE program on the boot disk and print the opening screen.

Using Nuts & Bolts to Save Setup Information

Using Nuts & Bolts, which comes on the CD accompanying this book, and following the directions in the chapter, create a rescue disk that contains a record of setup information.